Distributed Query Answering in Peer-to-peer Reasoning Systems

Technical Report

Arnold Binas and Sheila A. McIlraith
Department of Computer Science, University of Toronto
Toronto, Ontario, Canada
{abinas,sheila}@cs.toronto.edu

April 10, 2007

Abstract

Interest in distributed reasoning is gaining momentum due to the emergence of the Semantic Web. In this work, we address the problem of meaningfully answering queries to distributed propositional reasoners which may be mutually inconsistent. We define a peer-to-peer query answering framework in which peers may be prioritized. Our formal framework includes distributed entailment relations for both consistent and inconsistent systems. We develop decentralized algorithms for computing answers to a restricted class of queries according to distributed entailment and prove their soundness and completeness. A heuristic for more efficiently computing query answers from prioritized, inconsistent systems is also provided and its effectiveness demonstrated empirically. We furthermore investigate global closed-world reasoning in our framework and show how our techniques can be used to achieve global generalized closed-world query answering from local closed-world reasoners in a restricted class of distributed peer-to-peer query answering systems.

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Distributed Query Answering

Distributed reasoning with propositional logic reasoners is gaining momentum motivated by the Semantic Web and the stance that fully expressive languages are too intractable to reason with in distributed settings for practical purposes [37]. In peer-to-peer distributed reasoning, each peer hosts a local knowledge base and reasoner and is connected to other peers with which it shares a communication language. In the Semantic Web setting, each peer may represent an online service or information source. A user may want to query such peers, expecting an answer which relies on the knowledge and reasoning capability of the whole network of peers, rather than just the peers being queried. Query answering in such a distributed setting requires fully decentralized algorithms. Algorithms for consequence-finding in such a peer-to-peer setting already exist [4]. However, these algorithms largely assume that different peers' knowledge bases are mutually consistent. In the Semantic Web setting with no centralized control over the contents of individual peers' knowledge bases, this is a very strong assumption. Work has been done on answering queries to several relational databases that might be mutually inconsistent or from peer-to-peer data sources with no reasoning capabilities [17, 15]. There has also been some preliminary work on consequence-finding in an inconsistent propositional peer-to-peer reasoning setting [20]. However, no robust distributed query answering algorithm for peer-to-peer propositional reasoning exists.

1.2 Contribution of this Report

The goal of the work presented in this report is to formalize a peer-to-peer query answering system that allows for mutually inconsistent peer knowledge bases. We aim to formally specify how inconsistency-tolerant query answering in such a system should look like and develop distributed algorithms to compute query answers in a decentralized fashion. In order to further disambiguate inconsistent knowledge, provide for a mechanism to handle trust, and ease the computation of query answers, we introduce priorities and incorporate priority handling into the algorithms. Lastly, we give examples of how global reasoning with local closed-world reasoners could look like and show how our techniques can be used to achieve meaningful global closed-world reasoning in a restricted case.

1.3 Overview

The rest of this report is structured as follows.

Section 2: Background introduces existing work on distributed reasoning as well as reasoning with inconsistencies. This section sets the stage for describing our work and communicates some important concepts necessary to read the rest of this report.

Section 3: Peer-to-peer Query Answering Systems gives a formal framework of a peer-to-peer reasoning system for which we later develop algorithms to solve the query answering problem. This section also defines how entailment, and thus query answering, should look like in our system.

Section 4: Query Answering in a PQAS tackles the query answering problem first for systems with mutually consistent peers and then generalizes to potentially inconsistent systems. Algorithms for query answering in a peer-to-peer reasoning system are given and their properties investigated and proved.

Section 5: Global Closed-world Reasoning in PQASs gives two examples of coherent global reasoning in a system in which local closed-world reasoners participate and discusses general policies for different kinds of global closed-world reasoning. This section furthermore demonstrates how our techniques can be used to achieve global generalized closed-world reasoning in a restricted class of peer-to-peer query query answering systems in which all local reasoners make the closed-world assumption.

Section 6: Implementation and Experiments discusses an implementation of our framework and an empirical evaluation of some of our techniques.

Section 7: Related Work examines previous work in related areas. We look at related work in distributed reasoning, reasoning with inconsistencies, trust in peer-to-peer or otherwise distributed systems, as well as global closed-world reasoning with multiple databases or reasoners in the Semantic Web setting.

Section 8: Conclusion summarizes the contributions of this report and discusses its limitations and resulting opportunities for future research.

2 Background

Distributed reasoning techniques have been investigated both as a means to reason more efficiently with large, individual knowledge bases as well as in truly distributed peer-to-peer settings. Reasoning with inconsistencies has also received some attention through several approaches in the centralized setting. In this report, we aim to achieve reasoning with inconsistencies in a distributed, peer-to-peer environment. In preparation, we will examine some of the most relevant work in both of these areas in this section.

2.1 Distributed Reasoning

Reasoning with distributed knowledge has been previously investigated for at least two purposes. For an individual, but large centralized knowledge base, distributed reasoning techniques can yield more efficient reasoning when splitting up the KB into components and reasoning within and between them. Amir and McIlraith's work on partition-based logical reasoning addresses this issue [6]. Secondly, the initial setting, such as in the Semantic Web, might already involve separate, but connected peer-to-peer reasoners. A user may now want to be able to reason with the combined knowledge and reasoning capability provided by several of these peers. Adjiman et al. provide a framework and algorithms for distributed consequence-finding in such a peer-to-peer inference system [4]. This line of work focuses mainly on consistent systems, although more recent work by Chatalic et al. provides for inconsistency handling to some limited extent [20]. We discuss both partition-based reasoning and peer-to-peer consequence-finding below.

2.1.1 Partition-based logical reasoning

In [6], Amir and McIlraith develop a set of algorithms for efficiently reasoning with large knowledge bases. In their approach, a large KB is partitioned into loosely connected clusters of related knowledge. The result of this partitioning is a partition graph, in which individual partitions are connected if they share symbols of their respective signatures. The edges between partitions are labeled with the intersection of both partitions' signatures; i.e. edge labels specify *link languages*. A partition graph is then converted into a tree by deleting certain edges and expanding the labels of others. Given such a tree partitioning of the global theory, a message-passing algorithm is devised for propagating inferences between partitions in order to solve a given query. Both forward and backward message-passing algorithms are introduced, where the latter also requires the partition graph to be a chain for completeness. A procedure to convert a partition graph into a chain is provided.

Partition-based logical reasoning (PBLR) is similar to fully distributed reasoning in that it answers a global query by combining knowledge from local KB partitions in a distributed fashion. However, those partitions and their connectivity are created as part of the process. In particular, given a large, originally individual knowledge base, partition-based reasoning assumes that we can manipulate the partition graph and turn it into a tree, which the message-passing algorithm requires for completeness. For obvious reasons, this is not possible in the general distributed reasoning setting where peers ("partitions") and their connectivity are fixed and cannot be manipulated by the user posing the query or the query answering mechanism itself. Thus the message-passing algorithms in [6] cannot be used in the general setting, where a graph might not (and most likely will not) be a tree.

2.1.2 Consequence-finding in peer-to-peer inferences systems

Adjiman et al. introduced the first consequence-finding algorithm for a peer-to-peer reasoning setting [4]. The authors consider a network of peer reasoners of propositional logic which are connected as acquaintances. Similarly to the partition-based reasoning work of Section 2.1.1, each pair of connected peers shares a set of variables, however relaxing the assumption that any two peers sharing vocabulary are actually connected by an edge labeled by *all* variables that appear in both peers' vocabularies. The peers whose knowledge bases contain some of the same variables thus may or may not share an edge for each of those variables. This is a suitable generalization to be made for a peer-to-peer setting, in which there is no global force connecting all peers that might share vocabulary. Furthermore, no particular structure (such as a tree) is assumed for the acquaintance graph. In such a peer-to-peer inference system (P2PIS), the consequence-finding problem is defined as finding the proper prime implicates of a given clause as implied by the system as a whole. Only prime implicates in a given target language are computed, i.e. only those consequences containing only variables from a given set of target variables. The algorithm is initially defined recursively and then transformed into a set of fully distributed message-passing procedures. Adjiman et al.'s algorithm is anytime and delivers consequences from more and more distant peers as its execution progresses. It is shown to be complete if all variables shared between any two peers are allowed to propagate between those peers. The authors demonstrate the scalability of their algorithm empirically, running it on their SOMEWHERE architecture on up to 1000 peers with randomly generated theories and queries.

We will now explain schematically Adjiman's recursive consequence-finding algorithm (RCF) as it shall serve as the basis for our algorithms in Section 4 and is therefore essential to understand for reading this report. Algorithm 1 gives a high-level view of the algorithm, which takes a literal and a peer as the input and returns a set of prime implicates given the entire peer-to-peer reasoning system. The algorithm is called recursively on the individual literals of local consequence clauses and the per-literal consequences combined back together to yield the consequences of the whole clauses. The user interested in the consequences of the single-literal query q starting in peer P issues the top-level call RCF(q, P), which in turn calls the recursive algorithm with history, RCFH. The history keeps track of the literals and peers in which a consequence was generated for each recursive call of RCFH. The initial history is empty.

For each recursive call to RCFH on a literal q, the algorithm first checks whether the history already contains the negation of the literal (line 2). If this is the case, a contradiction in this branch of the search is detected and a set containing the empty clause ($\{\Box\}$) is returned. If q is already contained in some peer P of the set SP, there are no new consequences as a result of the query and the empty set is returned (line 4). Likewise, if q already occurs in the history with all of the peers in SP, i.e. has been processed by all of the current peers before, a cycle has been detected and the empty set is returned since all new consequences have already been found in the previous rounds of processing of q (line 4).

If neither an immediate contradiction nor a cycle are found, the consequences of q local to the peers in SP are computed by resolution and stored in the set LOCAL (line 6). If the set of local consequences includes the empty clause, no further acquainted peers have to be consulted and a set containing the empty clause consequence is returned (line 7). As Adjiman's algorithm searches only for clauses in a given target language, all local consequences whose non-shared literals are not in the target language are then thrown out (line 9). Next, the algorithm checks whether any of the clauses contain literals that are shared with acquainted peers and if not returns the

Algorithm 1 Adjiman's recursive consequence-finding algorithm for consistent P2PISs

```
1: **RCF**$(q, P)$
2: **return** RCFH$(q, \{P\}, \emptyset)$

1: **RCFH**$(q, SP, hist)$
2: **if** $\neg q$ occurs in $hist$ **then**
3:     **return** $\{\square\}$
4: **if** for some $P \in SP$, $q \in P$ or $q$ occurs in $hist$ for all $P \in SP$ **then**
5:     **return** $\emptyset$
6: LOCAL $\leftarrow \{q\} \cup (\bigcup_{P \in SP} Resolvent(q, P))$
7: **if** $\square \in$ LOCAL **then**
8:     **return** $\{\square\}$
9: LOCAL $\leftarrow \{c \in$ LOCAL$|$ all non-shared literals of $c$ are in the target language$\}$
10: **if** for all $c \in$ LOCAL, none of $c$'s literals are shared **then**
11:     **return** LOCAL
12: RESULT $\leftarrow$ LOCAL
13: **for all** $c \in$ LOCAL s.t. at least one of $c$'s literals is shared **do**
14:     let $P$ be the peer of $SP$ s.t. $c \in Resolvent(q, P)$
15:     **for all** shared literals $l \in c$ **do**
16:         ANSWER$(l) \leftarrow$ RCFH$(l, ACQ(l, P), [(q, P, c)|hist])$
17:     DISJCOMB $\leftarrow \otimes_{l\ shared}$ANSWER$(l) \otimes$ (unshared literals of $c$)
18:     RESULT $\leftarrow$ RESULT $\cup$ DISJCOMB
19: **return** RESULT
```

local consequences right away (line 10). If there are clauses with shared literals, for each such literal l of each such clause c, the consequences in the set of acquainted peers ($ACQ(l, P)$) are computed recursively (line 16). For each literal l, the set of consequences is stored in ANSWER(l). To form the consequences for the clause c from the consequences for it shared literals l, the distributed disjunction of the per-shared-literal consequences and the unshared literals is taken (line 17). $\otimes$ is the distributed disjunction operator that forms the Cartesian product of the sets of consequences it is given and disjoins them. Once all remote consequences for all clauses with shared literals have been found recursively, the set of local and remote consequences is returned as the set of consequences for q (line 19).

The recursive algorithm is then distributed by the authors into the message-passing algorithm DeCA (DEcentralized Consequence-finding Algorithm) to find consequences of a literal in a given target language in a real-world distributed peer-to-peer consequence-finding setting.

2.2 Reasoning with Inconsistencies

Reasoning with inconsistencies is the second theme of this report. In this subsection we introduce some of the most relevant work on reasoning with inconsistencies. Arenas and Bertossi et al. developed techniques to give consistent query answers from inconsistent databases [7, 14]. Benferhat et al. compared and evaluated several entailment relations for inconsistent knowledge bases, both for prioritized and non-prioritized KBs [11, 12]. Chatalic et al. build on Adjiman et al.'s peer-to-peer inference systems work to devise a set of algorithms for dealing with peer-to-peer inference systems in which the peers may be mutually inconsistent [20]. In the following, we discuss all three lines of work.

2.2.1 Inconsistent databases and integrated data sources

In this line of work, Arenas and Bertossi et al. consider the task of query answering in databases that might violate certain integrity constraints. Integrity constraints are first-order formulas over database relations. In [7], Arenas et al. define a database instance r to be inconsistent if it does not satisfy all integrity constrains. A repair of a database instance is another instance that (i) satisfies the integrity constraints and (ii) is closest to the original instance in terms of the relations it satisfies. A consistent query answer is then defined to be a ground tuple in a database instance all of whose repairs satisfy the query. The goal is to compute consistent answers to queries in database instances that may violate some of the integrity constraints. In the approach of Arenas and Bertossi et al., a query is rewritten into a query whose regular answers are the consistent answers of the original query [7, 14]. The authors introduce a method for doing so and prove its soundness, completeness for some classes of queries and integrity constraints, as well as its termination.

In [17], Bertossi et al. use and extend these techniques to achieve consistent query answers from multiple, integrated data sources. The authors develop an approach to derive query plans to gather information from separate databases seen as views of a global schema in a way that ensures global consistency. Again, a query to the global schema is rewritten into a query whose regular answers are the consistent answers of the original query. A query plan to query local data sources is then derived from the rewritten global query and executed. As opposed to pre-existing methods for building query plans, the authors' method deals with negative queries which arise after rewriting to ensure the consistency of the answers. Bertossi and Chomicki give a detailed account of query answering in inconsistent databases in [16].

2.2.2 Handling inconsistent knowledge bases

Benferhat et al. published a set of papers which are concerned with drawing inferences from inconsistent knowledge bases [11, 12]. Two different settings are investigated: the non-prioritized (or flat) case as well as prioritized knowledge bases. For each of the two settings, the authors define a series of entailment relations and compare their respective merits and drawbacks. Below we summarize their work.

Flat knowledge bases Benferhat et al. define several types of entailment in individual, inconsistent knowledge bases [11]. All are based on the concept of a maximally consistent subset of the (inconsistent) theory. A maximally consistent subset is a consistent subset of the theory, no superset (within the theory) of which is also consistent. The entailment relations are as follows. A *free-consequence* is one that is classically entailed from a set of formulas, none of which belongs to any minimally inconsistent subset of the theory. A minimally inconsistent subset is a set of formulas of the theory, the removal of any one of which would make the set consistent. An *MC-consequence* is a formula that is entailed by each maximally consistent subset of the theory. An *argued consequence* is one which is entailed by at least one consistent subset of the theory, and whose negation is entailed by no consistent subset of the theory. The riskiest entailment relation defined in this work is *existential-consequence* [36]. A formula is an existential-consequence if it is entailed by at least one maximally consistent subset of the theory. This is computationally relatively easy to determine, however, note that the negation of an existentially entailed formula may also be existentially entailed at the same time. Computation with the other entailment relations can be prohibitively hard as the number of maximally consistent subsets of a theory grows exponentially with the number of clashes (inconsistencies) within that theory. Thus computing all such subsets is hard in general. To address this issue, the authors define a relaxation of MC-consequence termed *L-consequence*. An L-consequence is a formula that is entailed by all maximally consistent subsets of maximal cardinality. All entailment relations considered by the authors are shown to be equivalent to classical logical entailment if applied to consistent knowledge bases.

Prioritized knowledge bases In related work, Benferhat et al. develop and compare several types of entailment for inconsistent knowledge bases whose formulas have priorities associated with them [13, 12]. In this work, a *prioritized knowledge base* Σ is seen as $\Sigma = S_1 \cup S_2 \cup \cdots \cup S_n$, the union of n *layers* of sets of formulas with those in S_1 having the highest and those in S_n having the lowest priority. The simplest (and least productive) entailment relation discussed by the authors is *possibilistic entailment*, or π-entailment ($\vdash_\pi$) based on possibilistic logic [43]. $\Sigma \vdash_\pi \phi$ iff there exists an $1 \leq i \leq n$ such that $\Sigma \vdash_i \phi$. $\Sigma \vdash_i \phi$ iff $\Sigma = S_1 \cup \cdots \cup S_i$ is consistent, entails ϕ, and for no $j < i$, $\Sigma' = S_1 \cup \cdots \cup S_j$ entails ϕ. Notice that this entailment relation suppresses some inferences that should reasonably be made by "drowning" formulas in S_k for $k > i$, even if those formulas would not cause inconsistencies if considered [10]. A more productive inference is *non-defeated consequence*. Let $Free(S)$ be the set of formulas in S which belong to no minimal inconsistent subset of S. The *dominant subbase* Σ^* of a prioritized knowledge base Σ is defined as $\Sigma^* = Free(S_1) \cup Free(S_2) \cup \cdots \cup Free(S_n)$. ϕ is a non-defeated inference iff $\Sigma^* \vdash \phi$. Every π-consequence is also a non-defeated one, and the converse is not true. The authors go on to describe the *RFS-consequence* relation. The yet more productive RFS-consequence relation is more laborious to define. Roughly speaking, $RFS(\Sigma)$ is the original knowledge base Σ with formulas involved in minimal inconsistent subbases iteratively removed in order of "positive influence" (see [13] for details) until it is consistent. A formula classically entailed by $RFS(\Sigma)$ is then said to be an RFS-consequence. One can argue that RFS-entailment is too productive as it can produce questionable results.

Benferhat et al. also introduce a number of entailment relations based on maximal consistent (MC) subsets of the theory. A strongly maximal consistent (SMC) subset is formed by starting with a MC subset of S_1, adding as many formulas from S_2 while preserving consistency, and so forth. An *SMC-consequence* is then any formula entailed by *all* SMC subbases. It turns out that the intersection of all SMC subbases contains the dominant subbase Σ^* and so SMC-entailment is more productive than non-defeated entailment. As with the MC-consequence relation for flat (non-prioritized) knowledge bases, the practicality of SMC-entailment is questionable due to the fact that the number of SMC subbases rises exponentially with the number of conflicts in the knowledge base. Analogous to the L-consequence for flat KBs, the authors introduce *Lex-consequence* for prioritized ones. See [13] for how $Lex(\Sigma)$ is defined as a subset of $SMC(\Sigma)$. Benferhat et al. also introduce a relation analogous to argued entailment for flat KBs. A *reason* A for a formula ϕ is a minimal consistent subbase that entails ϕ. The rank of a reason is the highest (weakest priority) non-empty layer of A. Thus the higher the rank of a reason the weaker it is. An *argued consequence* of Σ is then a formula ϕ for which a reason exists and all reasons for $\neg\phi$ are of higher rank. It is shown that argued entailment subsumes π-entailment. It is also shown that while argued entailment will never produce directly contradictory results (ϕ and $\neg\phi$), a set of multiple argued consequences of the same knowledge base may be inconsistent.

Lastly, the authors introduce a series of entailment relations that are based on what they call the defeasibility and safety of reasons. A reason A's defeasibility is the rank of the best reason for the most weakly supported belief in A. Its safety is the rank of the best reason for the most strongly attacked formula in A. An *SD-consequence* is one that is supported by a reason of minimal defeasibility within the set of reasons of maximal safety, and whose defeasibility is less than its safety. A *DS-consequence* is one that is supported by a reason of maximal safety within the set of reasons of minimal defeasibility, and whose defeasibility is less than its safety. An *SS-consequence* is one that is supported by any reason whose defeasibility is less than its safety. DS-consequences are a superset of π-consequences, while not all π-consequences are also SD-consequences. The SS-consequence relation is more productive than the DS-consequence relation but less productive than the non-defeated consequence.

The authors conclude that argued and SS-consequence are the most appealing due to the fact that they do not reduce the original knowledge base to one or multiple consistent subsets to draw inferences from but rather weigh pros and cons for and against inferring a formula. Although argued entailment can result in a set of consequences being inconsistent, Benferhat et al. argue that it is the most suitable as it is the only one being fully faithful to the actual contents of the knowledge base. It is argued that while a set of consequences may be inconsistent, the reasons supporting them are distinct and could be interpreted as simply different points of view. Based on this judgment and the fact that for the work presented in this report, we are concerned with a single query at a time, we shall adopt and adapt argued entailment for our work in Section 3.

2.2.3 Reasoning with inconsistencies in P2PISs

Building on the work on peer-to-peer reasoning systems (P2PISs) that we discussed in Section 2.1.2, Chatalic et al. introduced an approach for finding *well-founded consequences* in a P2PIS in which peers can be mutually inconsistent [20]. To the best of our knowledge, this is the only peer-to-peer distributed logical reasoning work to date that attempts to handle inconsistencies. According to the authors, a well-founded consequence is one which is based on a consistent subset of the global theory. The authors modify the specification of a P2PIS to include mapping clauses between individual peers. A mapping clause is a clause which contains variables of two peers, and all interactions between peers happen through mapping clauses. Given the assumption that individual peers' theories are consistent and any inconsistencies arise by combining individual peer theories, it is clear that any inconsistency must involve at least one mapping clause. The authors define a *nogood* as a set of mapping clauses leading to an inconsistency. A complete algorithm computing nogoods at the time of a new peer joining the network is provided. The algorithm is based on the DeCA algorithm developed in [4] by searching for the empty clause and keeping track of the different ways it can be derived. Given that each nogood in the system is stored by some peer in the network, the authors then provide an algorithm (again based on DeCA) which generates consequences, only reporting those that can be proved using sets of mappings not involving a nogood.

There is one limitation in this approach that we address in the present report, is that the notion of entailment used by Chatalic et al. (well-founded consequences) is quite limited and allows for both a formula and its negation to be produced as well-founded consequences, making it unsuitable for query answering. We shall address this issue in our work by using a different definition for entailment in the inconsistent setting.

3 Peer-to-peer Query Answering Systems

In this section we set the stage for solving the distributed query answering problem in propositional logic by formalizing a peer-to-peer query answering system and defining entailment relations for it. We will distinguish between consistent and inconsistent systems. In the next section, we will provide actual query answering algorithms for some classes of the peer-to-peer query answering system defined here.

3.1 Framework and Definitions

A *peer-to-peer query answering system (PQAS)* is a collection of interconnected peers, each of which hosts a local reasoner with a local knowledge base. Each reasoner produces consequences according to its local consequence relation. Reasoners are able to take in new clauses from other, connected peers and consider them in their reasoning process. All formulas, both those residing within local knowledge bases as well as those produced by the consequence relations, are in the language of propositional logic with bottom ($\perp$). In order to resolve potential inconsistencies, establish a model of trust, and ease the computation of query answers, a priority ordering is imposed over the peers. For the purposes of this work, we define priorities as drawn from a totally ordered set. The set may be a set of numeric values or it may be qualitative (e.g. {excellent, good, average, poor}). In the following, we define the necessary terms and concepts more thoroughly.

Definition 3.1 (Priorities)**.** *A priority is an attribute drawn from a possibly infinite, totally ordered set. Priorities are drawn from such a set with repetition, so that a priority can be either equal to or different from other priorities. The priority I_i is higher than priority I_j iff $I_i < I_j$, lower than I_j iff $I_i > I_j$, and equal to I_j otherwise. Higher priorities are more desirable than lower priorities. A set Σ of priorities can be aggregated into a single overall priority, or rank, by taking the maximum of the individual priorities' values: $max_{I \in \Sigma}(I)$. Thus the rank of a set of priorities is equal to the lowest priority in the set.*

Note that a priority is considered higher than another priority if the value of the former (e.g. a numeric value if the set they are drawn from is a set of numbers) is less than that of the latter. Without loss of generality, we will assume for the rest of this report that priorities are given as integer numbers. In Section 8.2 we briefly discuss other priority systems that may be possible.

Definition 3.2 (Peer-to-peer query answering system (PQAS))**.** *A peer-to-peer query answering system $\mathcal{P}$ is a tuple (P, G) where $P = \{P_i\}_{i=1}^n$ is a set of n peers and G is a graph (V, E) describing the communication connections between those peers. A peer P_i is a triple $(KB_i, Cons_i, I_i)$ comprising a local propositional knowledge base KB_i, its local consequence relation $Cons_i$, and the peer's priority I_i which adheres to Definition 3.1 above. We call $\bigcup_{i=1}^n KB_i$ the global theory of $\mathcal{P}$. We use the notation $\Sigma \models_i \phi$ to denote $\phi \in Cons_i(\Sigma)$, where Σ is a set of formulas. We say that peer P_i has higher priority than peer P_j iff $I_i < I_j$. A peer's signature L_i is the set of non-logical symbols in its knowledge base. Each peer's signature L_i is a subset of the global signature $L = \bigcup_{j=1}^n L_j$ of $\mathcal{P}$. V is the set $\{1, \ldots, n\}$ of vertices in G, with vertex i corresponding to peer P_i. E is the set of labeled edges (i, j, L_{ij}) in G, where L_{ij} is the edge label (link signature) between peers P_i and P_j and $L_{ij} \subseteq L_i \cap L_j$.*

A peer's local KB_i is an arbitrary propositional theory without any syntactic restrictions. We will briefly discuss possible extensions to the first-order case in Section 8.2 with respect to future work. Each $Cons_i$ is an arbitrary consequence relation which may or may not be classical. For example, one, several, or all peers might locally perform closed-world reasoning ($Cons_{CWA}$), in which non-provable atoms are assumed false. Thus neither the form of the local knowledge base KB_i nor the local consequence relation $Cons_i$ need to be the same for all i, beyond all local knowledge bases being propositional. The graph G describes the connectivity between peers. Two peers P_i and P_j that are connected by an edge (i, j, L_{ij}) can exchange consequences whose signature is in L_{ij}. Should consequences be contradictory, those from peer P_i will take precedence over those from P_j iff P_i has higher priority than P_j, i.e. iff $I_i < I_j$. An exception may be any consequences resulting from a local closed-world assumption, which we will discuss more in Section 5. Note that imposing priorities on our peers is not a restriction. A non-prioritized PQAS can be modeled by setting all peers' priorities to a single, fixed value.

Example 3.3. *Figure 1 shows a simple example of a PQAS. Each peer P_i is represented as a box containing its local knowledge base KB_i, which defines its signature L_i. Each peer also has a priority I_i associated with it. Peers are connected by edges which are labeled by the link signatures, which are subsets* *of the intersection of the two peers' respective signatures that are connected by the edge.*

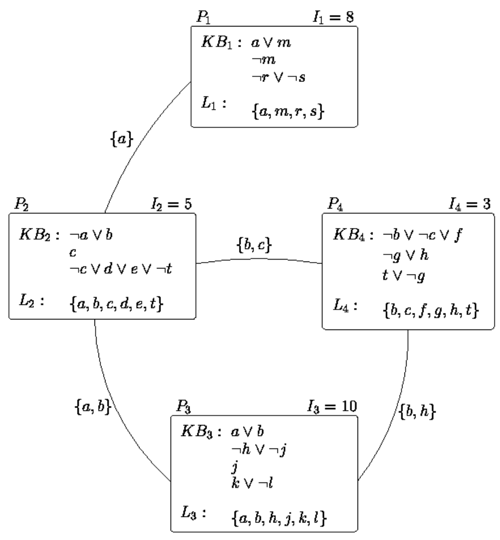

Figure 1: A simple PQAS

A PQAS $\mathcal{P}$ can be such that conflicts can or cannot arise. We call the former situation *$\mathcal{P}$-inconsistency* and the latter *$\mathcal{P}$-consistency*. I.e. a PQAS $\mathcal{P}$ is $\mathcal{P}$-consistent iff its reasoners never produce contradictory results. The concept of $\mathcal{P}$-consistency is defined more formally below.

Definition 3.4 ($\mathcal{P}$-consistency)**.** *Let $\mathcal{P} = (P, G)$ be a PQAS. Let each $P_i \in P$ generate all consequences of its local knowledge base KB_i according to its local consequence relation $Cons_i$ and for each edge $(i, j, L_{ij}) \in E$, let P_i pass all its local consequences whose signature is in L_{ij} to the connected peer P_j. Let each peer P_i now find the consequences of its local knowledge base KB_i in conjunction with all consequences received from other, connected peers, and again pass on all resulting consequences to neighboring peers according to the edges in E. Let this process repeat indefinitely. If $\perp$ is ever generated as a consequence in any peer in $\mathcal{P}$ during this process, $\mathcal{P}$ is said to be $\mathcal{P}$-inconsistent. It is said to be $\mathcal{P}$-consistent otherwise.*

The following proposition shows that in the special case of a PQAS with only classical local entailment relations and a graph that allows all consequences to propagate between peers, $\mathcal{P}$-consistency is equivalent to the satisfiability of the global theory.

Proposition 3.5. *Let $\mathcal{P} = (P, G)$ be a PQAS such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, $Cons_i$ is the classical entailment relation $\models$. Then $\mathcal{P}$ is $\mathcal{P}$-consistent iff $\bigcup_{i=1}^{n} KB_i$ is satisfiable. Otherwise it is $\mathcal{P}$-inconsistent.*

Proof. $\Rightarrow$: Assume $\mathcal{P}$ is $\mathcal{P}$-consistent. We will show that $\bigcup_{i=1}^{n} KB_i$ is satisfiable. By the definition of $\mathcal{P}$-consistency, $\perp$ cannot be produced from the consequences of peers in $\mathcal{P}$ that are allowed to propagate between them. By the assumption that whenever for two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$, *all* consequences are allowed to propagate between peers. Hence $\bigcup_{i=1}^{n} KB_i \not\models \perp$, which is equivalent to stating that $\bigcup_{i=1}^{n} KB_i$ is satisfiable.

$\Leftarrow$: Assume that $\bigcup_{i=1}^{n} KB_i$ is satisfiable. We will show that $\mathcal{P}$ is $\mathcal{P}$-consistent. Since the set is satisfiable, the consequences of $\bigcup_{i=1}^{n} KB_i$ cannot be used to generate $\perp$. This is exactly the definition of $\mathcal{P}$-consistency and thus $\mathcal{P}$ is $\mathcal{P}$-consistent. ■

3.2 Entailment in a $\mathcal{P}$-consistent PQAS

Recall that our motivation is to reason with distributed peers. In the following, we characterize at the knowledge level what distributed entailment of an arbitrary propositional formula in a PQAS should look like. We first look at a $\mathcal{P}$-consistent system. In such a system, no inconsistencies can occur during consequence generation and exchange between the peers. Therefore, entailment in a $\mathcal{P}$-consistent PQAS is defined as the existence of a reasoning sequence through a collection of peers that produces a formula ϕ. Recall that two peers can only pass a formula between each other if the graph of the PQAS contains an edge between the two peers that is labeled by a superset of the formula's signature. Since there are no inconsistencies to be resolved, any priority ordering over the peers is not needed to resolve conflicts. However, priorities can be used in a $\mathcal{P}$-consistent system for the computation of a *best reason* to believe a query. In the following, we recursively define distributed entailment in a $\mathcal{P}$-consistent PQAS as well as the concept of a best-rank reason.

Definition 3.6 (Distributed entailment in a $\mathcal{P}$-consistent PQAS ($\models_D$)). *ϕ is entailed by $\mathcal{P}$ by distributed entailment, denoted $\mathcal{P} \models_D \phi$, iff there exists a peer P_i s.t. $\mathcal{P} \models_i^D \phi$. $\mathcal{P} \models_i^D \phi$ iff one of the following two conditions holds: (1) $KB_i \models_i \phi$ or (2) there exists a set of formulas $\psi_1, \ldots, \psi_m$ s.t. $KB_i \cup \bigcup_{j=1}^m \{\psi_j\} \models_i \phi$ and for each ψ_j, there exists a peer P_k s.t. $\mathcal{P} \models_k^D \psi_j$ and $(i, k, L_{ij}) \in E$ where $L_{ij} \supseteq sig(\psi_j)$ and $sig(\psi_j)$ is the signature of ψ_j.*

Notice that for $\mathcal{P}$ to entail a formula ϕ by distributed entailment, the existence of a peer P_i is required such that ϕ is either derived in P_i alone (condition (1)), or P_i aggregates and reasons with formulas received from connected peers to derive ϕ (condition (2)). In the latter case, P_i may either be a peer of $\mathcal{P}$ or a an additional *query peer* which is connected to some peers of $\mathcal{P}$ and whose purpose is to ask a given query. To see why this may be necessary, imagine, for example, the conjunctive query $s \wedge t$ and a PQAS $\mathcal{P}$ in which no single peers' local signature includes both s and t. Then no peer of the system will be able to locally derive $s \wedge t$ from any intermediate formulas derived by other, connected peers. A query peer asking $s \wedge t$, however, will be connected to both the peers that include s in the local signature as well as those that include t. Naturally, the query peer then includes both s and t in its local signature and may derive $s \wedge t$ from the individual formulas s and t that it receives from other, connected peers.

The following proposition shows that in the special case of a $\mathcal{P}$-consistent PQAS with only classical local entailment relations and a graph that allows all consequences to propagate between peers, distributed entailment of a formula in the PQAS is equivalent to its classical entailment by the global theory.

Proposition 3.7. *Let $\mathcal{P} = (P, G)$ be a $\mathcal{P}$-consistent PQAS such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, $Cons_i$ is the classical entailment relation $\models$. Then $\mathcal{P} \models_D \phi$ iff $\bigcup_{i=1}^n KB_i \models \phi$.*

Proof. $\Rightarrow$: Let ϕ be an arbitrary formula such that $\mathcal{P} \models_D \phi$. By the definition of $\models_D$, this means that there exists a peer P_k s.t. $\mathcal{P} \models_k^D \phi$. We will show that $\bigcup_{i=1}^n KB_i \models \phi$ by induction on the number of recursive applications of Definition 3.6.

Base case (no recursive applications). Since there are no recursive applications of the definition, condition (1) of the definition holds and

$KB_k \models_k \phi$. Then clearly $\bigcup_{i=1}^n KB_i \models \phi$ since $\models_k$ is the classical entailment relation $\models$ and $KB_k \in \bigcup_{i=1}^n KB_i$.

Induction step (r recursive applications). Assume that for $r' < r$ recursive applications of the definition of $\models_k^D$ the induction hypothesis is true and if $\mathcal{P} \models_s^D \psi$ then $\bigcup_{i=1}^n KB_i \models \phi$. Let $\mathcal{P} \models_k^D \phi$ require r applications of the definition of $\models_k^D$.

In this case, condition (2) of the definition holds and for some set of m formulas $\{\psi_j\}_{j=1}^m$, $KB_k \cup \bigcup_{j=1}^m \{\psi_j\} \models_k \phi$ and for each ψ_j, there exists a peer P_s s.t. $\mathcal{P} \models_s^D \psi_j$. By the induction hypothesis, for all ψ_j, $\bigcup_{i=1}^n KB_i \models \psi_j$. Since $KB_k \cup \bigcup_{j=1}^m \{\psi_j\} \models_k \phi$, $KB_k \in \bigcup_{i=1}^n KB_i$, and $\models_k$ is the classical entailment relation, we have $\bigcup_{i=1}^n KB_i \models \phi$ and are done.

$\Leftarrow$: Let ϕ be an arbitrary formula such that $\bigcup_{i=1}^n KB_i \models \phi$. We will show that $\mathcal{P} \models_D \phi$. $\bigcup_{i=1}^n KB_i \models \phi$ means that for some k, $\phi \in Cons_k(\Sigma)$ for some set of formulas $\Sigma \in \bigcup_{i=1}^n KB_i$. The formulas in Σ are either all in $Cons_k(KB_k)$, which is equivalent to case (1) of Definition 3.6 and therefore $\mathcal{P} \models_D \phi$. Or the formulas in Σ are formulas in $Cons_k(KB_k)$ and additionally m formulas $\{\psi_j\}_{j=1}^m$ that are entailed by $\bigcup_{i=1}^n KB_i$. Assuming that for each j, $\mathcal{P} \models_D \psi_j$, and given that all formulas in P_k's signature can be propagated to P_k (since for all non-empty L_{ij}, an edge between P_i and P_j labeled by L_{ij} exists), this corresponds to case (2) of Definition 3.6. Now each ψ_j is recursively shown to be entailed by distributed entailment in $\mathcal{P}$, so we are done. The recursion must end at some point because there are only finitely many formulas classically entailed by $\bigcup_{i=1}^n KB_i$. ■

Given that the formula ϕ is entailed by a PQAS $\mathcal{P}$, i.e. at least one reason for ϕ in $\mathcal{P}$ exists, it makes sense to talk about a best-rank reason if multiple reasons exist. To define the best-rank reason, first the concept of a reason in a PQAS as well as the rank of a reason must be formally defined.

Definition 3.8 (Reason for a formula). *Given a PQAS $\mathcal{P}$ and a formula ϕ, if $\mathcal{P} \models_D \phi$, then a set of formulas $\Sigma \in \bigcup_{i=1}^n KB_i$ is a reason for ϕ iff Σ contains all and only the axioms involved in a derivation of ϕ in $\mathcal{P}$.*

Definition 3.9 (Rank of a reason). *The rank of a reason Σ, denoted $R(\Sigma)$, is the priority of the lowest-priority formula in it. That is, $R(\Sigma) = max_{\psi \in \Sigma}(prio(\psi))$, where the priority $prio(\psi)$ of a formula ψ is the priority I_i of the peer P_i it originated from. Analogous to priorities, a rank is said to be higher than another rank if the value of the former is less than that of the latter, lower* *than another rank if the value of the former is greater than that of the latter, and the two ranks are said to be equal otherwise.*

I.e. the rank of a reason is the priority of the lowest-priority peer contributing a formula to the reason. Recall that a reason Σ only contains the original formulas from the local knowledge bases involved in the reason and no formulas derived thereof and therefore every formula in a reason originates from some peer's local knowledge base KB_i.

Definition 3.10 (Best-rank reason)**.** *Among all reasons for a formula ϕ with respect to a $\mathcal{P}$-consistent PQAS $\mathcal{P}$, the best-rank reason is the reason with the highest rank.*

I.e. the best-rank reason for a formula has the lowest priority value among all ranks for the formula. The best-rank reason for a formula may be of interest in a PQAS where priorities represent the reliability or trustworthiness of peers.

3.3 Entailment in a $\mathcal{P}$-inconsistent PQAS

We now turn to the more general case of entailment in a $\mathcal{P}$-inconsistent PQAS. We employ here the concepts of *reasons* and *argued entailment* as introduced by Benferhat et al. in [13] in order to draw consequences from inconsistent formula sets. We reviewed this work in Section 2.2.2. In their work, Benferhat et al. define an argued consequence from a prioritized set of formulas as one that is entailed from a consistent subset of those formulas which has higher rank than any consistent subset entailing the negated consequence. The rank of a formula set is a measure for the combined priority of the formulas in it. Adapted to our peer-to-peer reasoning framework, this means the following. Similarly as in the $\mathcal{P}$-consistent case, a reason in support of a formula ϕ is a minimal set of original formulas from one or multiple peers that derives ϕ, respecting the graph of the PQAS which may restrict which formulas can be sent between which peers. Additionally, in the $\mathcal{P}$-inconsistent case the formula set constituting a reason is explicitly required to be consistent. Again, the rank of a reason is the priority of the source peer of the lowest-priority formula that occurs in the reason. A formula ϕ is said to be entailed by the PQAS if there exists a reason for ϕ and all reasons for $\neg\phi$ are of lower rank. We formalize these concepts in the following definitions.

Definition 3.11 (Support of a formula by a reason)**.** *The formula ϕ is supported in a $\mathcal{P}$-inconsistent PQAS by the reason Σ, denoted $\mathcal{P} \models_{\Sigma} \phi$, iff there exists a peer P_i s.t. $\mathcal{P} \models_i^{\Sigma} \phi$. $\mathcal{P} \models_i^{\Sigma} \phi$ iff $\Sigma \not\models \bot$ and one of the following conditions holds: (1) $\Sigma \subseteq KB_i$, $\Sigma \models_i \phi$, and for no $\Sigma^* \subset \Sigma$, $\Sigma^* \models_i \phi$ or (2) there exists a set of formulas $\Sigma' \subseteq KB_i$ and a set of formulas $\psi_1, \ldots, \psi_m$ s.t. $\Sigma' \cup \bigcup_{j=1}^{m}\{\psi_j\} \models_i \phi$, for no $\Sigma'^* \subset \Sigma'$, $\Sigma'^* \cup \bigcup_{j=1}^{m}\{\psi_j\} \models_i \phi$, and for each ψ_j, there exists a peer P_k and a set of formulas Σ_j s.t. $\mathcal{P} \models_k^{\Sigma_j} \psi_j$ and $(i, k, L_{ij}) \in E$ with $L_{ij} \supseteq sig(\psi_j)$, and $\Sigma = \Sigma' \cup \bigcup_{j=1}^{m} \Sigma_j$.*

Analogously to the definition of distributed entailment for $\mathcal{P}$-consistent PQASs, the definition of a reason above may require the involvement of a query peer (P_i in condition (2) of Definition 3.11). The rank $R(\Sigma)$ of a reason Σ is defined as in Definition 3.9 for the $\mathcal{P}$-consistent case.

Definition 3.12 (Distributed entailment in a prioritized PQAS ($\models_D$) (following [13]))**.** *A formula ϕ is entailed by the PQAS $\mathcal{P}$ iff there exists a reason Σ^+ supporting ϕ and for all reasons Σ^- supporting $\neg\phi$, $R(\Sigma^+) < R(\Sigma^-)$.*

Notice that a priority ordering can be partial or total. In the case of a partial priority ordering of the peers, it is possible that an inconsistency cannot be resolved. For example, ϕ and $\neg\phi$ might be produced by two different peers with the same lowest priority. In this case, neither ϕ nor $\neg\phi$ would be concluded by the system. However, in a system with a total priority ordering over the peers, an inconsistency ϕ and $\neg\phi$ cannot be resolved only if the lowest-priority peer in the reason for ϕ and the lowest-priority peer in the reason for $\neg\phi$ are the same peer.

4 Query Answering in a PQAS

In this we discuss query answering in a peer-to-peer query answering system according to the entailment relations introduced in the previous section. A query is asked by a query peer which is either one of the peers in the PQAS or an additional peer with its own signature connected to some of the peers in the PQAS. More specifically, the query answering problem is defined as follows.

Definition 4.1 (Query answering problem in a PQAS)**.** *The query answering problem in a PQAS is to pose a formula in the language of the query peer and determine whether it is* true*,* false*, or* unknown *given the combined knowledge of the entire PQAS.*

We will focus here on a restricted class of queries for PQASs in which all peers use classical entailment as their local consequence relation. The types of queries we will develop query answering algorithms for in this section are single-literal, conjunctive, and some disjunctive queries.

At the core of any method that solves the query answering problem is an algorithm solving single-literal queries. The answer to a conjunctive query can be trivially constructed from the answers to its single-literal queries. A conjunctive query succeeds iff all of its single-literal subqueries succeed. While many disjunctive queries can be handled by a similar decomposition into subqueries, some disjunctive queries prove much more difficult to handle. For simplicity, we will start by looking at single-literal queries only.

Again we will focus on the two different cases for which specific algorithms can be developed, namely $\mathcal{P}$-consistent and $\mathcal{P}$-inconsistent PQAS. The following subsection discusses single-literal query answering algorithms for $\mathcal{P}$-consistent PQASs, and Section 4.2 discusses algorithms for the $\mathcal{P}$-inconsistent case. For both situations, we first describe recursive algorithms and then distribute them into message-passing procedures similarly to [4]. We will extend the algorithms to handle conjunctive queries and discuss disjunctive queries in Section 4.3. Finally, Section 4.2.6 briefly talks about finding not only an answer, but also the best-rank reason for it in $\mathcal{P}$-consistent systems.

To summarize, the present section makes the following assumptions.

- For each peer P_i of the PQAS $\mathcal{P}$, the peer's local consequence relation $Cons_i$ is classical entailment.

- A query is asked by a query peer that may be one of the peers of $\mathcal{P}$, or an additional peer connected to some of the peers in $\mathcal{P}$.
- In Sections 4.1 and 4.2, a query is a single literal q.

4.1 $\mathcal{P}$-consistent PQASs

Recall that in a $\mathcal{P}$-consistent system no inconsistent consequences are produced and we therefore need not consider any priority ordering to resolve such inconsistencies when trying to answer a query. Priorities can still be used to compute the best-rank reason for a query, discussed more in Section 4.2.6, but we will focus in this subsection on determining whether a query is true or false, i.e. whether a reason exists, regardless of its rank. Recall that for the purpose of this subsection, each peer's local consequence relation is the classical entailment relation and a query is a single, positive or negative literal. Several distributed reasoning algorithms already exist. We mention the two most relevant here. Amir and McIlraith's partition-based reasoning work produced a query answering algorithm which could be used for $\mathcal{P}$-consistent PQAS whose graphs are trees [6]. This case, however, is severely restrictive, and we shall not consider it in more detail here. A consequence-finding algorithm for consequences of single literals in what we call $\mathcal{P}$-consistent systems with arbitrary graphs also already exists in Adjiman et al.'s work [4]. Adjiman's consequence-finding algorithm is explained in detail as Algorithm 1 in Section 2.1.2 on peer-to-peer inference systems. We can use Adjiman's consequence-finding algorithm to do query answering by posing the negation of our query to see if a contradiction is derived as a consequence. As we operate in a $\mathcal{P}$-consistent system, any contradiction can only be caused by the negated query. In this subsection, we develop this special case of Adjiman's algorithm first as a recursive and then as a message-passing algorithm. We furthermore give an example to show the operation of the algorithms and discuss some of their more important properties.

4.1.1 Recursive query answering algorithm

A recursive algorithm is developed before a distributed one for conceptual purposes. Our query answering adaptation of Adjiman's recursive algorithm is shown as Algorithm 2. We modify Adjiman's algorithm to fix the target language to the empty language (since we are only looking for the empty clause $\Box$) and to terminate immediately (by recursively returning $true$) as soon as the empty clause is found. $true$ represents the empty clause being a consequence of the given literal, while $false$ represents the literal having no consequences in the empty target language. The algorithm produces consequences locally in the query peer, and for each neighboring peer and each local clause whose literals are all shared, the clause is split into its individual literals and the algorithm called recursively for each. If for any local consequence c, the consequences for each of its literals in other peers contain the empty clause ($true$ is returned), then the empty clause is also a consequence of c and $true$ is returned in the current call. If Adjiman_QA receives $true$ from the recursive algorithm Adjiman_QAR (also in Algorithm 2) then YES is returned as the answer to the original query, otherwise NO is returned. In Section 4.1.4 we show that Algorithm 2 is a special case of Adjiman's consequence-finding algorithm for P2PISs and which properties therefore apply.

The following is some notation needed to read Algorithm 2 as well as all further algorithms discussed in this section.

Definition 4.1 (History (following [4])). *A history hist is a list of tuples (l, P, c) of a literal l, a peer P, and a clause c. c is a consequence of l in P, and l is a literal of the clause of the previous tuple in the hist list.*

Definition 4.2 (Local consequences (following [4])). *$Resolvent(l, P_i)$ is the set of local consequences in peer $P_i = (KB_i, Cons_i, I_i)$ of the literal l, according to its local consequence relation $Cons_i$. I.e. $Resolvent(l, P_i) = \{c | KB_i \models_i c \vee \neg l\}$.*

Definition 4.3 (Acquainted peers by literal (following [4])). *$ACQ(l, P)$ is the set of all peers which share an edge labeled by L_{ij} with peer P, where $l \in L_{ij}$. I.e. $ACQ(l, P) = \{P' \in V | (P, P', L_{ij}) \in E \text{ and } l \in L_{ij}\}$.*

Algorithm 2 Query answering adaptation of Adjiman's algorithm for $\mathcal{P}$-consistent PQASs

```
1: Adjiman_QA($q, P$)
2: if Adjiman_QAR($\neg q, \{P\}, \emptyset$) then
3:    return $YES$
4: else
5:    return $NO$

1: Adjiman_QAR($p, SP, hist$)
2: if $(\neg p, \_, \_) \in hist$ then
3:    return $true$
4: if there exists $P \in SP$ s.t. $p \in P$ or $\forall P \in SP, (p, P, \_) \in hist$ then
5:    return $false$
6: LOCAL $\leftarrow \{p\} \cup (\bigcup_{P \in SP} Resolvent(p, P))$
7: if $\Box \in$ LOCAL then
8:    return $true$
9: LOCAL $\leftarrow \{c \in$ LOCAL$|$ all literals in $c$ are shared$\}$
10: for all $c \in$ LOCAL do
11:    let $P$ be the peer of $SP$ s.t. $c \in Resolvent(p, P)$
12:    for all literals $l \in c$ do
13:       ANSWER($l$) $\leftarrow$ Adjiman_QAR($l, ACQ(l, P), [(p, P, c)|hist]$)
14:    if $\bigwedge_{l \in c}$ANSWER($l$) $= true$ then
15:       return $true$
16: return $false$
```

4.1.2 Message-passing query answering algorithm

In this subsection we turn the recursive Algorithm 2 into a message-passing algorithm along the lines of Adjiman's DeCA algorithm described in [4] and in lesser detail in Section 2.1.2 of this report. There are three message types that peers can send to each other and one procedure to handle each of them. *Forth* messages are sent for each literal for which the sender peer requests the computation of consequences by its neighbors in order to look for the empty clause $\square$. *Back* messages are sent to the sender peer if $\square$ has been found to be a consequence of the respective literal. *Final* messages indicate that the exploration of a particular search branch has been completed. More precisely, a *final* message is sent back to the sender peer iff the empty clause has been found to be a consequence or a cycle in the message-passing history has been detected (i.e. the search has "been there" before). The three message-passing procedures are shown in Algorithms 3, 4, and 5. A new query q is posed by the $User$ to query peer P by sending the message $\mathrm{m}(User, P, forth, \emptyset, \neg q)$ and succeeds when receiving a message $\mathrm{m}(P, User, back, hist)$. If the query cannot be proved by the system, the $User$ is eventually notified by a message $\mathrm{m}(P, User, final, hist)$ without receiving a corresponding *back* message. We will show in Section 4.1.4 that the message-passing algorithm computes the same results as the recursive one, and therefore enjoys the same properties.

4.1.3 Example

Here we will conceptually demonstrate the operation of our message-passing algorithm by the means of an example. Consider again the simple PQAS we looked at in Section 3, reproduced here as Figure 2. We have four peers P_i, each with a different priority I_i. Each peer has a local knowledge base KB_i which induces a local signature L_i. The edges between the peers are labeled with the corresponding link signatures, which are subsets of the intersections of the signatures of the two peers connected by the edge. Each local consequence relation $Cons_i$ is the classical consequence relation adhering to classical entailment ($\models$). Notice that our example PQAS is $\mathcal{P}$-consistent as the global theory ($\bigcup_i KB_i$) is consistent.

In this setting, let us consider a user peer, which is connected to P_4, posing the query f. Our algorithm first negates the query and passes it to P_4 for consequence-finding. The progression of one possible proof is shown in Figure 3. The underlined clauses are the ones split by the algorithm where the

Algorithm 3 Query answering version of Adjiman's *forth* message algorithm

```
1: ReceiveForthMessage(m($Sender, Self, forth, hist, p$))
2: if $(\neg p, \_, \_) \in hist$ then
3:    send m$(Self, Sender, back, [(p, Self, \square)|hist])$
4:    send m$(Self, Sender, final, [(p, Self, true)|hist])$
5: else if $p \in Self$ or $(p, Self, \_) \in hist$ then
6:    send m$(Self, Sender, final, [(p, Self, true)|hist])$
7: else
8:    LOCAL$(Self) \leftarrow \{p\} \cup Resolvent(p, Self)$
9:    if $\square \in$ LOCAL$(Self)$ then
10:       send m$(Self, Sender, back, [(p, Self, \square)|hist])$
11:       send m$(Self, Sender, final, [(p, Self, true)|hist])$
12:   else
13:       LOCAL$(Self) \leftarrow \{c \in$ LOCAL$(Self) \mid$ all literals in $c$ are shared$\}$
14:       if LOCAL$(Self) = \emptyset$ then
15:          send m$(Self, Sender, final, [(p, Self, true)|hist])$
16:       for all $c \in$ LOCAL$(Self)$ do
17:          for all $l \in c$ do
18:             BOTTOM$(l, [(p, Self, c)|hist]) \leftarrow false$
19:             for all $RP \in ACQ(l, Self)$ do
20:                FINAL$(l, [(p, Self, c)|hist], RP) \leftarrow false$
21:                send m$(Self, RP, forth, [(p, Self, c)|hist], l)$
```

Algorithm 4 Query answering version of Adjiman's *back* message algorithm

```
1: ReceiveBackMessage(m($Sender, Self, back, hist$))
2: $hist$ is of the form $[(l', Sender, c'), (p, Self, c)|hist']$
3: BOTTOM$(l', [(p, Self, c)|hist']) \leftarrow true$
4: if $\forall l \in c$, BOTTOM$(l, [(p, Self, c)|hist']) = true$ then
5:    if $hist' = \emptyset$ then
6:       $U \leftarrow User$
7:    else
8:       $U \leftarrow$ the first peer $P'$ of $hist'$
9:    send m$(Self, U, back, [(p, Self, c)|hist'])$
10:   send m$(Self, U, final, [(p, Self, c)|hist'])$
```

Algorithm 5 Query answering version of Adjiman's *final* message algorithm

```
1: ReceiveFinalMessage(m($Sender, Self, final, hist$))
2: $hist$ is of the form $[(l', Sender, true), (p, Self, c)|hist']$
3: FINAL($l', [(p, Self, c)|hist'], Sender$) $\leftarrow true$
4: if $\forall c^* \in$ LOCAL($Self$) and $\forall l \in c^*$, FINAL($l, [(p, Self, c^*)|hist'], \_$) $= true$
   then
5:     if $hist' = \emptyset$ then
6:         $U \leftarrow User$
7:     else
8:         $U \leftarrow$ the first peer $P'$ of $hist'$
9:     send m($Self, U, final, [(p, Self, true)|hist']$)
```

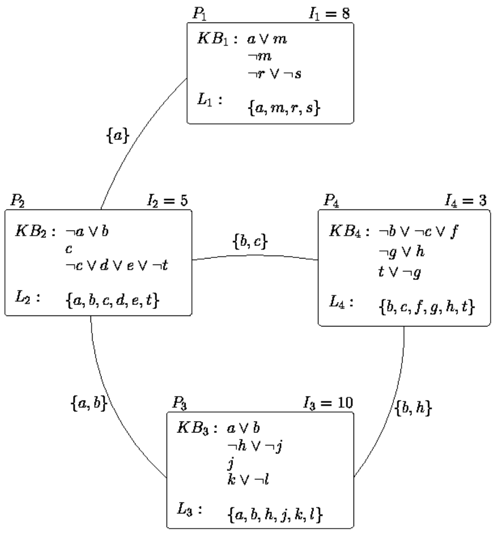

Figure 2: A $\mathcal{P}$-consistent PQAS

split-off literal generating consequences in a particular peer is underlined as well. Thick lines represent connections between peers along which messages are passed (recursive calls issued). A downward arrow represents a *forth* message and an upward arrow a *back* message. Within each peer, resolution is used among the local KB's clauses and the passed-in literal, where only those local clauses are shown which are used in the proof. The refutation proof makes use of the clauses $\neg f$, $\neg b \vee \neg c \vee f$, $\neg a \vee b$, $a \vee m$, $\neg m$, and c.

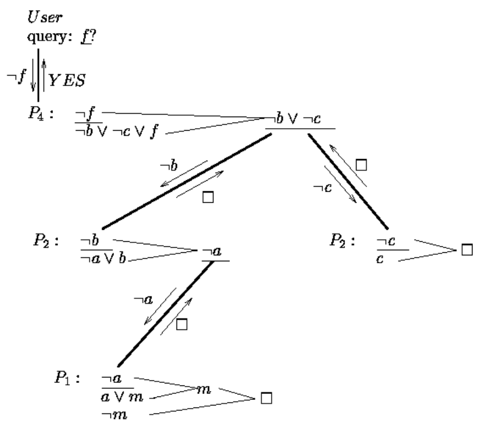

Figure 3: A proof of f

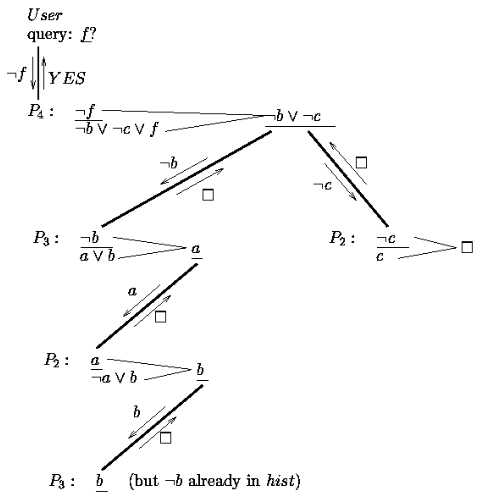

Figure 4: Another proof of f

The proof visualized in Figure 3 is not the only possible proof of the query f in our PQAS. Figure 4 shows another proof of the same query. In this case, the split-off literal $\neg b$ of the clause $\neg b \vee \neg c$ at the top leads to a contradiction not by the derivation of the empty clause directly, but by a history that contains the negated literal of a subquery, activating the check on line 2 of Algorithm 3. Consider the left branch of the proof tree. Consequences in various peers (P_4, P_3, and P_2) are generated from the following literals, in this order: $\neg f$, $\neg b$, and a. In peer P_2, a leads to the derivation of b, which is then passed on to P_3. Arriving at P_3, the literal b comes with the following history: $[(a, P_2, b), (\neg b, P_3, a), (\neg f, P_4, \neg b \vee \neg c)]$. Line 2 of Algorithm 3 will now fire since $(\neg b, \_, \_)$ is part of the history coming with literal b and *true* (signaling that the empty clause was found) is returned via a *back* message. To convince ourselves that the proof is indeed valid, we again look at the set of clauses that it makes use of to realize that it entails the empty clause: $\neg f, \neg b \vee \neg c \vee f, a \vee b, \neg a \vee b, c$.

4.1.4 Properties of the algorithms

In this subsection we discuss some properties of the query answering algorithms for $\mathcal{P}$-consistent PQASs. First, we will show that our recursive query answering algorithm is a special case of Adjiman's consequence-finding algorithm and therefore enjoys the same properties. Secondly, we will prove our algorithm's soundness and completeness with respect to distributed entailment for $\mathcal{P}$-consistent PQASs. We will then transfer those results to the message-passing algorithm by showing that it computes the same results as the recursive one.

Theorem 4.4 below states that our query answering algorithm in Algorithm 2 is a special case of Adjiman's recursive consequence-finding algorithm.

Theorem 4.4 (Restricted Adjiman algorithm)**.** *Given a PQAS $\mathcal{P}$, Adjiman_QA(q, P) of Algorithm 2 returns YES iff RCF(q, P) of Algorihtm 1 in Section 2.1.2 returns $\Box$ as one of the consequences.*

Proof. $\Rightarrow$: Assume Adjiman_QA(q, P) returns YES. We will show that RCF(q, P) returns $\Box$ as one of the consequences. Adjiman_QA(q, P) returns YES iff Adjiman_QAR$(\neg q, \{P\}, \emptyset)$ returns *true*. RCF(q, P) returns $\Box$ iff RCFH$(q, \{P\}, \emptyset)$ returns $\Box$. It remains to be shown that if Adjiman_QAR$(\neg q, \{P\}, hist)$ returns *true* then RCFH$(q, \{P\}, hist)$ returns $\Box$ as one of the consequences. Adjiman_QAR$(\neg q, \{P\}, hist)$ returns *true* if (i) $\neg p$ occurs in the history (line 2), (ii) the empty clause is locally derived (line 7), or (iii) for all literals l of a locally derived clause c, Adjiman_QAR$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns *true* (line 14).

(i) In this case, RCFH$(q, \{P\}, hist)$ returns $\Box$ (line 2).

(ii) In this case, RCFH$(q, \{P\}, hist)$ returns $\Box$ (line 7).

(iii) In this case, RCFH$(q, \{P\}, hist)$ returns $\Box$ iff
RCFH$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns $\Box$ for each $l \in c$ as the distributed disjunction of all consequences of the recursive calls is added to the set RESULT (line 17). RCFH$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns $\Box$ if Adjiman_QAR$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns *true* by a recursive argument.

$\Leftarrow$: Assume RCF(q, P) returns $\Box$ as one of the consequences. We will show that Adjiman_QA(q, P) returns YES. RCF(q, P) returns $\Box$ iff RCFH$(q, \{P\}, \emptyset)$ returns $\Box$. Adjiman_QA(q, P) returns YES iff Adjiman_QAR$(\neg q, \{P\}, \emptyset)$ returns *true*. It remains to be shown that if RCFH$(q, \{P\}, hist)$ returns $\Box$ as one of the consequences then Adjiman_QAR$(\neg q, \{P\}, hist)$ returns *true*.

RCFH$(\neg q, \{P\}, hist)$ returns $\Box$ as one of the consequences if (i) $\neg p$ occurs in the history (line 2), (ii) the empty clause is locally derived (line 7), or (iii) for all literals l of a locally derived clause c, RCFH$(\neg l, ACQ(l, P), [(p, P, c)|hist])$

returns $\Box$ as one of the consequences (lines 17 and 18 as the set RESULT is eventually returned).

(i) In this case, Adjiman_QAR$(q, \{P\}, hist)$ returns *true* (line 2).

(ii) In this case, Adjiman_QAR$(q, \{P\}, hist)$ returns *true* (line 7).

(iii) In this case, Adjiman_QAR$(q, \{P\}, hist)$ returns *true* iff Adjiman_QAR$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns *true* for each $l \in c$ as the conjunction of all return values of the recursive calls is *true* iff all return values for the recursive calls are *true* (line 14).
Adjiman_QAR$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns *true* if RCFH$(\neg l, ACQ(l, P), [(p, P, c)|hist])$ returns $\Box$ as one of the consequences by a recursive argument. ■

Since by Theorem 4.4, Algorithm 2 is a special case of Adjiman's recursive consequence-finding algorithm, the soundness, termination, and conditional completeness results with respect to *classical entailment* from the global theory in [4] apply without change. In particular, the algorithm is sound with respect to classical entailment and guaranteed to terminate. It is furthermore complete with respect to classical entailment under the condition that for each pair of peers and each variable that is in both their local vocabularies, a path between the two peers exists such that each edge of the path is labeled with the variable.

We are, however, also interested in the more general case of query answering with respect to distributed entailment from a PQAS. The algorithm is considered sound with respect to distributed entailment as defined in Definition 3.6 if whenever it returns YES, the original query is true. It is considered complete with respect to distributed entailment if it returns YES for every true query asked. In the following theorems, we show that Algorithm 2 is both sound and complete with respect to distributed entailment.

Theorem 4.5 (Soundness wrt. distributed entailment)**.** *Given a $\mathcal{P}$-consistent PQAS $\mathcal{P}$ and a peer P connected to it, if Adjiman_QA(q, P) of Algorithm 2 returns YES then $\mathcal{P} \models_D q$.*

Proof. P may be a peer in $\mathcal{P}$ or an additional query peer that is connected to peers in $\mathcal{P}$. Adjiman_QA(q, P) returns YES iff Adjiman_QAR$(\neg q, \{P\}, \emptyset)$ returns *true* by proof by refutation of the negated query. It is thus sufficient to show that if Adjiman_QAR$(\neg q, \{P\}, \emptyset)$ returns *true* then $\mathcal{P} \models_D q$. Hence we need to look at the recursive calls of the algorithm Adjiman_QAR$(p, SP, hist)$. The history *hist* has the form $[(h_n, P_n, c_n), \ldots, (h_1, P_1, c_1)]$

for $n \geq 0$, where the h_m are the literals that the derivation of p in the current reasoning branch depends on. It remains to be shown that if Adjiman_QAR$(p, SP, hist)$ returns $true$, then $\mathcal{P} \models_D \bigwedge_{m=1}^{n} h_m \to \neg p$, or $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \to \neg p$ for some peer P_i (by the definition of distributed entailment), which we will show by induction on the number of recursive calls the algorithm needs to terminate. Given a proof for this statement, the top-level call Adjiman_QAR$(\neg q, \{P\}, \emptyset)$ then implies $\mathcal{P} \models_D q$ since its history is empty.

To be shown: Adjiman_QAR$(p, SP, hist)$ returns $true \Rightarrow \mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \to \neg p$.

Base case (no recursive calls). $true$ is returned by the algorithm either (i) if $(\neg p, \_, \_) \in hist$ (line 2) or (ii) if the empty clause is a local consequence in some peer $P_i \in SP$ (line 7). **(i)** Clearly, $\neg p \to \neg p$, so $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \to \neg p$ since one of the $h_m = \neg p$. **(ii)** Since the empty clause is locally derived from p in a peer $P_i \in SP$ given the literals h_m from the history, we have

$$
\begin{aligned}
& KB_i \cup \{p\} \models_i \bigwedge_{m=1}^{n} h_m \to \bot \\
\Rightarrow \quad & KB_i \models_i \bigwedge_{m=1}^{n} h_m \to \neg p \\
\Rightarrow \quad & \mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \to \neg p.
\end{aligned}
$$

The last step is by condition (1) of Definition 3.6 with $\phi = \bigwedge_{m=1}^{n} h_m \to \neg p$.

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and if Adjiman_QAR$(l, SP', [(p, P_i, _)|hist])$ returns $true$ then there is a peer $P_j \in SP'$ s.t. $\mathcal{P} \models_j^D \bigwedge_{m=1}^{n} h_m \wedge p \to \neg l$. Let Adjiman_QAR$(p, SP, hist)$ be a call to the algorithm that requires r recursive calls to terminate.

Since the algorithm requires at least one recursive call to terminate, $true$ is returned if there exists clause c that is a local consequence of p in some peer $P_i \in SP$ and for all literals $l \in c$, Adjiman_QAR$(l, ACQ(l, P_i), [(p, P_i, _)|hist])$ returns $true$ (line 14). By the induction hypothesis, we have for each $l \in c$, $\mathcal{P} \models_j^D \bigwedge_{m=1}^{n} h_m \wedge p \to \neg l$ for some peer $P_j \in ACQ(l, P_i)$ (line 13) **(I)**. By Definition 4.3 of ACQ, $P_j \in ACQ(l, P_i)$ implies that $(i, j, L_{ij}) \in E$ where $l \in L_{ij}$ **(II)**. Consider now the extension of the local knowledge base KB_i by the set of clauses $\{\bigwedge_{m=1}^{n} \neg h_m \vee \neg p \vee \neg l\}_{l \in c}$. Then we have

$$\begin{aligned} & KB_i \cup \{\bigwedge_{m=1}^{n} \neg h_m \vee \neg p \vee \neg l\}_{l \in c} \models_i \bigwedge_{m=1}^{n} \neg h_m \vee \neg p \\ \Rightarrow\ & KB_i \cup \{\bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l\}_{l \in c} \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \ \ \textbf{(III)} \end{aligned}$$

where the entailment in the first line is true since $KB_i \models_i c$ and c resolves with the $\{\neg l\}_{l \in c}$ under the classical entailment relation $Cons_i$. The second line is obtained from the first one by rewriting the disjunctions on either side of the $\models_i$ as implications.

But **(I)**, **(II)**, and **(III)** are just condition (2) of Definition 3.6 with the $\psi_j = \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l$ for all $l \in c$ and $\phi = \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$. Therefore, by the definition of distributed entailment, $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$ and we are done. ■

Theorem 4.6 (Completeness wrt. distributed entailment)**.** *Given a $\mathcal{P}$-consistent PQAS $\mathcal{P}$ and a peer P connected to it, if $\mathcal{P} \models_D q$ then Adjiman_QA(q, P) of Algorithm 2 returns YES.*

Proof. P may be a peer in $\mathcal{P}$ or an additional query peer that is connected to peers in $\mathcal{P}$. Adjiman_QA(q, P) returns YES iff Adjiman_QAR($\neg q, \{P\}, \emptyset$) returns *true*. It is thus sufficient to show that Adjiman_QAR($\neg q, \{P\}, \emptyset$) returns *true* if $\mathcal{P} \models_D q$ or, by the definition of distributed entailment, if $\mathcal{P} \models_i^D q$ for some peer P_i.

Rewritten by substituting $\neg q$ with p, and generalized to include a possibly empty history *hist*, we need to show that if $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$ for some peer $P_i \in SP$ then Adjiman_QAR($p, SP, hist$) returns *true*. The history has the form $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1)]$ for $n \geq 0$, where the h_m are the literals that the derivation of p in the current reasoning branch depends on. We will show this by induction on the number of recursive applications of Definition 3.6 in $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$.

To be shown: $\mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \Rightarrow$ Adjiman_QAR($p, SP, hist$) returns *true*.

Base case (no recursive applications). In this case we have by condition (1) of the definition

$$\begin{aligned} & \mathcal{P} \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \\ \Rightarrow \quad & KB_i \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \\ \Rightarrow \quad & KB_i \cup \{\neg p\} \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \bot . \end{aligned}$$

If one of the $h_m = \neg p$, i.e. the above is trivially true, then $(\neg p, \_, \_) \in hist$ (line 2 of Algorithm 2) and $true$ is returned.

Otherwise, line 6 computes the set $\{\phi | KB_i \cup \{\neg q\} \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \phi\}$. By the assumption, the empty clause $\Box$ is in that set and the algorithm returns $true$ by the condition on line 7.

Induction step (r recursive applications). Assume that for $r' < r$ recursive applications of the definition the induction hypothesis is true and if there is a peer $P_j \in SP'$ s.t. $\mathcal{P} \models_j^D \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l$ then Adjiman_QAR$(l, SP', [(p, P_i, _)|hist])$ returns $true$. Let $KB_i \models_i^D \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$ require r recursive applications of the definition of $\models_i^D$.

Since at least one recursive application of the definition is required, condition (2) applies and we have

$$\begin{aligned} & KB_i \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \ \ (\mathbf{I}) \ \ and \\ & for\ each\ \psi_j\ \exists P_k\ s.t.\ \mathcal{P} \models_k^D \bigwedge_{m=1}^{n} h_m \rightarrow \psi_j\ and\ (i, k, L_{ik}) \in E\ where\ L_{ik} \supseteq sig(\psi_j) \ \ (\mathbf{II}). \end{aligned}$$

(I) can be rewritten as

$$KB_i \cup \{p\} \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \bot .$$

By refutation completeness and since KB_i is consistent, there must exist a clause c s.t.

$$\begin{aligned} & KB_i \cup \{p\} \models_i c \\ and \quad & \{c\} \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \bot . \end{aligned}$$

Since $\mathcal{P}$ is $\mathcal{P}$-consistent by assumption, the set $\{\psi_j\}_j$ is consistent and does not entail $\perp$. So $\perp$ must result from clauses in $\{\psi_j\}_j$ resolving with all literals $l \in c$. c is a descendant of the negation of the original query and as such must be involved in deriving $\perp$. The $\{\psi_j\}_j$ must thus entail $\neg l$ for each $l \in c$. Without loss of generality, we can thus assume that **(I)** holds and can be rewritten in the form

$$KB_i \cup \{\neg l\}_{l \in c} \models_i \bigwedge_{m=1}^{n} h_m \wedge p \to \perp.$$

Furthermore, **(II)** is of the form

$$\mathcal{P} \models_k^D \bigwedge_{m=1}^{n} h_m \wedge p \to \neg l$$

for some peer P_k for each $l \in c$. But for this case, Adjiman_QAR$(l, ACQ(l, P_i), [(p, P_i, _)|hist])$ of line 13 of Algorithm 2 returns *true* for each $l \in c$ by the induction hypothesis and since $P_k \in ACQ(l, P_i)$ because $(i, k, L_{ik}) \in E$ with $l \in L_{ij}$ by **(II)**. Since for each $l \in c$, *true* is returned by the recursive call, the algorithm Adjiman_QAR$(p, SP, hist)$ returns *true* on line 14 and we are done. ■

We have shown that Algorithm 2 is sound and complete with respect to distributed entailment. That is, Algorithm 2 returns YES iff the query is entailed by distributed entailment from the PQAS.

Consider the special case of a PQAS $\mathcal{P}$ with a graph that allows all consequences to propagate between peers. In this case, the algorithm is complete with respect to classical entailment from the global theory $\bigcup_{i=1}^{n} KB_i$. Corollary 4.7 below verifies this observation.

Corollary 4.7. *Let $\mathcal{P} = (P, G)$ be a $\mathcal{P}$-consistent PQAS such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, $Cons_i$ is the classical entailment relation $\models$. Algorithm 2 is complete with respect to classical entailment from the global theory $\bigcup_{i=1}^{n} KB_i$.*

Proof. Algorithm 2 is complete with respect to distributed entailment by Theorem 4.6. But under the assumptions of the corollary, distributed entailment in $\mathcal{P}$ is equivalent to classical entailment from the global theory by Proposition 3.7. Therefore, under the assumptions above, Algorithm 2 is complete with respect to classical entailment from the global theory $\bigcup_{i=1}^{n} KB_i$. ■

The following theorem verifies that Algorithm 2 is guaranteed to terminate, i.e. the user will eventually receive an answer for the query.

Theorem 4.8 (Termination (following [4])). *Algorithm 2 terminates.*

Proof. This proof is following that of termination of Adjiman's consequence-finding algorithm in [4]. For each recursive call, a *new* element is added to the history (because of the $(p, \_, \_) \in hist$ check on line 2). There are finitely many peers in the PQAS and finitely many literals and possible combinations thereof (i.e. clauses) per peer. An non-terminating algorithm would produce an infinite history, which is not possible due to the uniqueness of its elements and the fact that there is only a finite number of possible elements. Therefore, the algorithm must terminate. ■

In a real, distributed PQAS we cannot employ the recursive algorithm but will use the message-passing algorithm. Thus we are interested in ensuring soundness, completeness, and termination for the latter. The following theorem does so by showing that the message-passing algorithm computes the same results as the recursive one and notifies the user upon termination and thus shows that the same properties hold for the message-passing algorithm as for the recursive one. The proof of the theorem is somewhat following that of Theorem 3 in [4] and adjusted for the search of the empty clause rather than all consequences.

Theorem 4.9 (Correspondence between recursive and message-passing algorithms). *Adjiman_QAR($p, SP, hist$) of Algorithm 2 returns true iff for some peer $P' \in SP$, if a peer P sends the message $m(P, P', forth, hist, p)$ (Algorithm 3), it will receive a message $m(P', P, back, [(p, P', \Box)|hist])$ (Algorithm 4). Furthermore, P will eventually receive a message $m(P', P, final, hist)$ (Algorithm 5).*

Proof. **Equivalence of algorithms' results.** We will first prove by induction on the number of recursive calls of Algorithm 2 the equivalence of the results of both the recursive and the message-passing algorithms by showing that the peer P receives the message $m(P', P, back, [(p, P', \Box)|hist])$ from some peer $P' \in SP$ iff Adjiman_QAR($p, SP, hist$) returns *true*.

Base case (no recursive calls). Adjiman_QAR($p, SP, hist$) returns *true* if $(\neg p, \_, \_) \in hist$ (line 2) or if the empty clause is locally derived (line 7). The former case corresponds to line 2 of the *forth* message handling procedure Algorithm 3, in which case the message $m(P', P, back, [(p, P', \Box)|hist])$ is sent. The latter case corresponds to line 9 of Algorithm 3, in which case the *back* message is sent as well.

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and for all empty clauses returned by calls of the recursive algorithm a corresponding *back* message is sent by the message-passing algorithm. Let Adjiman_QAR($p, SP, hist$) be a call to the algorithm that requires r recursive calls to terminate.

Adjiman_QAR($p, SP, hist$) returns $true$ if for each literal of some local consequence, the algorithm recursively returns $true$ (line 14). Let $c = p_1 \vee \cdots \vee p_m$ be a local consequence of a peer P' such that for each p_i, a recursive call Adjiman_QAR($p_i, ACQ(p_i, P'), [(p, P', c)|hist]$) returns $true$ for a peer $P_i \in ACQ(p_i, P')$. By the induction hypothesis, a message $m(P_i, P', back, [(p_i, P_i, \square)|hist_i])$ is sent for each i. For each such message, Algorithm 4 records that literal p_i generated the *back* message (line 3). When a *back* message for the last of c's literals is received (line 4), the message $m(P', P, back, [(p, P', \square)|hist'])$ is sent since c was a local consequence of p in P'. This corresponds to the condition on line 14 of Algorithm 2.

Notification of termination. It remains to be shown that the message-passing algorithm notifies the query peer P of termination once it occurs. In other words, we need to show that once the empty clause has been shown to be a consequence, or it has been shown that it is not a consequence, of a given literal, a *final* message $m(P', P, final, hist)$ is sent. We will again proceed by induction.

Base case (no recursive calls). The empty clause is not a consequence of p in P' if either p is a clause of the local KB, $(p, \_, \_)$ already occurs in the history (i.e. we have a cycle and all potential consequences have been found on the first visit already), or $\square$ is not locally derived and there are no local consequences all of whose literals are shared. In the former case, a *final* message is sent on line 5 of Algorithm 3, and in the latter case a *final* message is sent on line 14.

Whenever the empty clause is derived, i.e. whenever a *back* message is sent, a *final* message is also sent at the same time as the empty target language allows for no more consequences than the empty clause and the branch that generated the empty clause can be closed.

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and whenever an empty clause is derived or it is established that it cannot be derived in the current branch, a *final* message is sent. Let Adjiman_QAR($p, SP, hist$) be a call to the algorithm that requires r recursive calls to terminate.

A *final* message needs to be sent once a *back* message is received for each literal in a local consequence as well as when a *final* message has been received for each literal in a local consequence. Line 10 in Algorithm 4 (condition on line 4) and line 9 in Algorithm 5 (condition on line 4) ensure that this is the case. ■

Theorem 4.9 shows that the message-passing algorithm computes the same results as the recursive algorithm and is therefore subject to the same properties. In particular, the message-passing algorithm is sound and conditionally complete with respect to classical entailment from the global theory. It is furthermore sound and complete with respect to distributed entailment. Lastly, the user is notified upon termination of the algorithm, which occurs when the empty clause is found in the successful case and once the accessible search space is exhausted in the case that the empty clause cannot be derived.

4.2 $\mathcal{P}$-inconsistent PQASs

In this subsection we present an algorithm which handles the most general case of a potentially $\mathcal{P}$-inconsistent PQAS with prioritized peers. We do so by adhering to distributed entailment ($\models_D$) as defined in Definition 3.12. I.e. both the query and its negation are asked and the one with the better-rank reason believed. Recall that in this subsection we again assume all local consequence relations to be classical and a query to be a single literal. As in the previous subsection, we will first develop a recursive algorithm and then derive a message-passing version before going through an example and discussing the algorithms' soundness, completeness, and termination properties.

4.2.1 Recursive query answering algorithm

The recursive algorithm is loosely modeled after Adjiman's recursive consequence-finding algorithm in [4] and our query answering adaptation of it shown as Algorithm 2 in the previous subsection. Our main contribution is the conversion to a query-answering algorithm, the handling of inconsistent global knowledge bases, and the introduction of priorities. Algorithms 6 and 7 show our query answering algorithm for the prioritized, $\mathcal{P}$-inconsistent setting. A call to Answer_Query initiates two calls to Find_Bottom, one of which attempts to prove the original query and one of which attempts to prove its negation, again both by refutation. Find_Bottom returns the set of all valid empty clauses found as a consequence of the query. We will see in a moment that stopping the search after the first empty clause found does not in general achieve coherent query answering in a $\mathcal{P}$-inconsistent system. Local consequences are again split as in the $\mathcal{P}$-consistent case and each of their literals' consequences found recursively in neighboring peers and then recombined. A found empty clause as a consequence of the negated original query indicates a reason for the query if all of its parent clauses other than the negation of the query form a consistent (satisfiable) set. A reason's rank is the weakest priority (highest numerical value) among all ancestor clauses of the empty clause representing a reason. The best-rank reasons for the query and its negation are selected and compared. If the query (or its negation) cannot be proved from a consistent set of clauses, the weakest possible rank (∞) is assumed. If the query can be proved with better rank than its negation, YES is returned. If the negation of the query can be proved with better rank than the original query, NO is returned. UNK (for unknown) is returned if both are proved with equal best-rank or both cannot be proved at all (rank ∞).

The recursive algorithm (Algorithm 7) acts similarly as the algorithm for the $\mathcal{P}$-consistent case in that it looks for an empty clause $\square$ in the system. However, Algorithm 7 does not return once an empty clause has been found but continues to search for other contradictions. For all algorithms and discussions in this subsection we use the notation $\square$ to mean *an* empty clause rather than *the* empty clause as each empty clause will represent a derivation thereof and will have different properties associated with it (an ancestor set and a priority as we shall see shortly). The reason for not stopping once the first derivation of an empty clause has been found is two-fold. Firstly, we are looking for the best-priority inconsistency arising from the negation of the query, not any-priority inconsistency. Secondly, since we operate in a $\mathcal{P}$-inconsistent system, any contradiction derived might come from an inconsistency already inherent in the system rather than from the interaction of the negation of the query and its positive derived in the system. In order to see if an empty clause arose from the negation of the query only, we need to keep track of the *original ancestors* of all clauses that went into the derivation of a particular instance of the empty clause, except the negation of the original query (which we need to be responsible for any "valid" contradictions derived). We call the set of original ancestors minus the negation of the query of a clause c $c.OA$. Any original clause in any of the peers' KBs has itself and itself only in its OA set. The negation of the query is an exception and has an empty OA set. When two clauses are resolved, the resolvent's OA set is the union of the two parent clauses' OA sets. The only exception is thus the negation of the query itself, which is an original ancestor to every clause ever derived by the algorithm, but not included in any clause's OA set. The negation of the query is left out of the OA sets so that a $\Box$ indicates a reason for the original query if $\Box.OA$[1] is satisfiable. If it is not satisfiable, then the empty clause resulted from an inconsistent subset of the clauses in the PQAS, and as such does not indicate an acceptable reason. An empty clause $\Box$ found to be a consequence of a literal l of a clause c is tested for satisfiability of its OA set as soon as it is discovered, and only re-combined with the consequences of the other literals in c if $\Box.OA$ is satisfiable.

The priority of a clause c is the lowest priority of any of its original ancestors (except the negation of the query) and denoted *c.prio*. The priority of an original ancestor is the priority of the peer it resides in. The priority I_i of a peer $P_i = (KB_i, Cons_i, I_i)$ is denoted P_i*.prio* in the algorithms. The rank of a reason is the priority of its corresponding empty clause and is denoted $\Box$*.prio*. The following additional piece of notation is also necessary to read the algorithms in this subsection.

Definition 4.10 (Distributed disjunction operator (following [4]))**.** *The distributed disjunction operator* $\otimes$ *in* $A \otimes B$ *forms clauses by the disjunction of all combinations of clauses in the sets* A *and* B *. For an indexed set of clause sets* $\{A_i\}_i$*, the notation* $\otimes_{i\in\{1,2,3\}} A_i$ *means* $A_1 \otimes A_2 \otimes A_3$*.*

[1]Again, in a slight abuse of notation, we denote by $\Box$ not the empty clause in general but an instance of the empty clause that differs from other instances by its properties $\Box.OA$ and $\Box$*.prio*.

Algorithm 6 Recursive query answering algorithm for $\mathcal{P}$-inconsistent PQASs

```
1:  **Answer_Query**($q, P$)
2:  $Pos \leftarrow$ Find_Bottom($\neg q, \{P\}, \emptyset$)
3:  $Neg \leftarrow$ Find_Bottom($q, \{P\}, \emptyset$)
4:  if $Pos = \emptyset$ then
5:      $pos \leftarrow \infty$
6:  else
7:      $pos \leftarrow \min_{\Box \in Pos}(\Box.prio)$
8:  if $Neg = \emptyset$ then
9:      $neg \leftarrow \infty$
10: else
11:     $neg \leftarrow \min_{\Box \in Neg}(\Box.prio)$
12: if $pos < neg$ then
13:     return $YES$
14: if $neg < pos$ then
15:     return $NO$
16: return $UNK$
```

Algorithm 7 Recursive query answering algorithm for $\mathcal{P}$-inconsistent PQASs

```
1: **Find_Bottom**($p, SP, hist$)
2: if $\forall P \in SP, (p, P, \_) \in hist$ then
3:     return $\emptyset$
4: RESULT $\leftarrow \emptyset$
5: if $(\neg p, \_, \_) \in hist$ then
6:     $hist$ is of the form $[(l', \_, c')|hist']$
7:     if $c'.OA$ is SAT then
8:         let $\Box$ be a new empty clause
9:         $\Box.OA \leftarrow c'.OA$
10:        $\Box.prio \leftarrow c'.prio$
11:        RESULT $\leftarrow$ RESULT $\cup\{\Box\}$
12: LOCAL $\leftarrow \{p\} \cup (\bigcup_{P \in SP} Resolvent(p, P))$
13: for all $c \in$ LOCAL do
14:     let $\{c_i^*\}_i$ be the set of clauses that went into $c$
15:     $c.OA \leftarrow \bigcup_i c_i^*.OA$ (recursively)
16:     $c.prio \leftarrow \max_i(c_i^*.prio)$ (recursively)
17: for all empty clauses $\Box \in$ LOCAL s.t. $\Box.OA$ is SAT do
18:     RESULT $\leftarrow$ RESULT $\cup\{\Box\}$
19: LOCAL $\leftarrow \{c \in \text{LOCAL}|c \neq \Box$, all literals in $c$ are shared$\}$
20: for all $c \in$ LOCAL do
21:     let $P$ be the peer of $SP$ s.t. $c \in Resolvent(p, P)$
22:     for all literals $l \in c$ do
23:         ANSWER($l$) $\leftarrow$ Find_Bottom($l, ACQ(l, P), [(p, P, c)|hist]$)
24:     DISJCOMB $\leftarrow \otimes_{l \in c}$ANSWER($l$)
25:     DISJCOMB $\leftarrow \{\Box \in$DISJCOMB$\}$
26:     for all $\Box \in$ DISJCOMB do
27:         for all literals $l_i \in c$ do
28:             let $\Box_i$ be the consequence of $l_i$ that went into $\Box$
29:         $\Box.prio \leftarrow \max_i(\Box_i.prio)$
30:         $\Box.OA \leftarrow \bigcup_i \Box_i.OA$
31:     DISJCOMB $\leftarrow \{\Box \in$ DISJCOMB$|\Box.OA$ is SAT$\}$
32:     RESULT $\leftarrow$ RESULT $\cup$ DISJCOMB
33: return RESULT
```

4.2.2 Message-passing query answering algorithm

The message-passing procedures for the $\mathcal{P}$-inconsistent case look similar to the ones for the $\mathcal{P}$-consistent case. However, there are a few important differences. Firstly, the message-passing algorithms introduced in this subsection do not terminate once an empty clause has been found. This is because we are looking for the *best-priority* empty clause that is based on a *satisfiable* set of clauses. Thus, whenever an empty clause is discovered, the satisfiability of its OA set is tested and no *final* message is sent (as opposed to the $\mathcal{P}$-consistent case where the first empty clause found will result in a *final* message). Notice one optimization in the message-passing version of this algorithm over the recursive version. A local variable *best* is kept at each peer to keep track of the priority of the best currently known reason for the query or its negation. *best* is initialized to infinity and lowered through the query peer (and only the query peer) when it receives an empty clause with satisfiable OA set and higher priority (corresponding to a reason with higher rank) than the current value of *best*. An additional message type, the *prio* message, is introduced to spread this new best-reason rank value to all peers in the network, which will then cut off any search branch and throw out any local consequences with lower priority (higher numeric value) than *best*. In order to determine the answer of a new query q using this message-passing algorithm, the query peer $User$ sends two messages: $\mathrm{m}(User, P, forth, \emptyset, q)$ and $\mathrm{m}(User, P, forth, \emptyset, \neg q)$, one each for the query and its negation. It will then receive a collection of *back* messages of the forms $\mathrm{m}(P, User, back, hist, \Box^{+})$ and $\mathrm{m}(P, User, back, hist, \Box^{-})$. Upon the receipt of a *final* message for both the query and its negation, the priorities of the highest-priority $\Box^{+}$ and $\Box^{-}$ are compared and the answer corresponding to the better value believed. We will prove in Section 4.2.4 that this strategy finds the same best-rank reasons for the original query and its negation as the recursive algorithm while being able to prune part of the search space.

Algorithm 8 *Forth* message algorithm for $\mathcal{P}$-inconsistent PQASs

```
1: ReceiveForthMessage(m($Sender, Self, forth, hist, p$))
2: if $(p, Self, \_) \in hist$ then
3:    send m($Self, Sender, final, [(p, Self, true)|hist]$)
4: else if $p.prio > best$ or $Self.prio > best$ then
5:    send m($Self, Sender, final, [(p, Self, true)|hist]$)
6: else
7:    if $(\neg p, \_, \_) \in hist$ then
8:       $hist$ is of the form $[(l', \_, c')|hist']$
9:       if $c'.OA$ is SAT then
10:         let $\Box$ be a new empty clause
11:         $\Box.OA \leftarrow c'.OA$
12:         $\Box.prio \leftarrow c'.prio$
13:         send m($Self, Sender, back, [(p, Self, \Box)|hist], \Box$)
14:         if $hist = \emptyset$ and $\Box.prio < best$ then
15:            send m($Self, Sender, prio, \Box.prio$)
16:    LOCAL($Self$) $\leftarrow \{p\} \cup Resolvent(p, Self)$
17:    for all $c \in$ LOCAL($Self$) do
18:       let $\{c_i^*\}_i$ be the set of clauses that went into $c$
19:       $c.OA \leftarrow \bigcup_i c_i^*.OA$ (recursively)
20:       $c.prio \leftarrow \max_i(c_i^*.prio)$ (recursively)
21:    LOCAL($Self$) $\leftarrow \{c \in$LOCAL($Self$)$|c.prio \leq best\}$
22:    $temp\_min \leftarrow \infty$
23:    for all $\Box \in$ LOCAL($Self$) s.t. $\Box.OA$ is SAT do
24:       send m($Self, Sender, back, [(p, Self, \Box)|hist], \Box$)
25:       $temp\_min \leftarrow \min(temp\_min, \Box.prio)$
26:    if $hist = \emptyset$ and $temp\_min < best$ then
27:       send m($Self, Sender, prio, temp\_min$)
28:    LOCAL($Self$) $\leftarrow \{c \in$ LOCAL($Self$) $|c \neq \Box$, all literals in $c$ are shared$\}$
29:    if LOCAL($Self$) $= \emptyset$ then
30:       send m($Self, Sender, final, [(p, Self, true)|hist]$)
31:    for all $c \in$ LOCAL($Self$) do
32:       for all $l \in c$ do
33:          CONS($l, [(p, Self, c)|hist]$) $\leftarrow \emptyset$
34:          $ACQ^* \leftarrow \{P' \in ACQ(l, Self)|P'.prio \leq best\}$
35:          for all $RP \in ACQ^*$ do
36:             FINAL($l, [(p, Self, c)|hist], RP$) $\leftarrow false$
37:             send m($Self, RP, forth, [(p, Self, c)|hist], l$)
38:    if no forth message sent then
39:       send m($Self, Sender, final, [(p, Self, true)|hist]$)
```

Algorithm 9 *Back* message algorithm for $\mathcal{P}$-inconsistent PQASs

```
1: **ReceiveBackMessage(m(**$Sender, Self, back, hist, \square^*$**))**
2: $hist$ is of the form $[(l', Sender, c'), (p, Self, c)|hist']$
3: CONS($l', [(p, Self, c)|hist']$) $\leftarrow$ CONS($l', [(p, Self, c)|hist']$) $\cup\ \square^*$
4: RESULT $\leftarrow (\otimes_{l \in c \setminus \{l'\}}$CONS$(l, [(p, Self, c)|hist'])) \otimes \{\square^*\}$
5: **for all** $\square \in$ RESULT s.t. $\square$ contains an empty clause consequence for each $l \in c$ **do**
6:     let $\{\square_i^*\}_i$ be the set of empty clauses that went into $\square$
7:     $\square.OA \leftarrow \bigcup_i \square_i^*.OA$
8:     $\square.prio \leftarrow \max_i(\square_i^*.prio)$
9: RESULT $\leftarrow \{\square \in$ RESULT$|\square.prio \leq best$ and $\square.OA$ is SAT$\}$
10: **if** $hist' = \emptyset$ **then**
11:     $U \leftarrow User$
12: **else**
13:     $U \leftarrow$ the first peer $P'$ of $hist'$
14: $temp\_min \leftarrow \infty$
15: **for all** $\square' \in$ RESULT **do**
16:     send m($Self, U, back, [(p, Self, c)|hist'], \square'$)
17:     $temp\_min \leftarrow$ min($temp\_min, \square'.prio$)
18: **if** $hist = \emptyset$ and $temp\_min < best$ **then**
19:     send m($Self, Sender, prio, temp\_min$)
```

Algorithm 10 *Final* message algorithm for $\mathcal{P}$-inconsistent PQASs

```
1: **ReceiveFinalMessage(m(**$Sender, Self, final, hist$**))**
2: $hist$ is of the form $[(l', Sender, true), (p, Self, c)|hist']$
3: FINAL($l', [(p, Self, c)|hist'], Sender$) $\leftarrow true$
4: **if** $\forall c^* \in$ LOCAL($Self$) and $\forall l \in c^*$, FINAL($l, [(p, Self, c^*)|hist'], \_$) $= true$ **then**
5:     **if** $hist' = \emptyset$ **then**
6:         $U \leftarrow User$
7:     **else**
8:         $U \leftarrow$ the first peer $P'$ of $hist'$
9:     send m($Self, U, final, [(p, Self, true)|hist']$)
```

Algorithm 11 *Prio* message algorithm for $\mathcal{P}$-inconsistent PQASs

```
1: ReceivePrioMessage(m($Sender, Self, prio, x$))
2: if $x < best$ then
3:     $best \leftarrow x$
4:     for all $RP \in ACQ(\_, Self)$ s.t. $RP \neq Sender$ do
5:         m($Self, RP, prio, best$)
```

4.2.3 Example

We will again demonstrate the operation of the message-passing algorithm by looking at an example. Figure 5 shows a modification of our earlier example to turn it into a $\mathcal{P}$-inconsistent PQAS. Notice that the clause $\neg x$ in peer P_2 is inconsistent with the clause x in peer P_4.

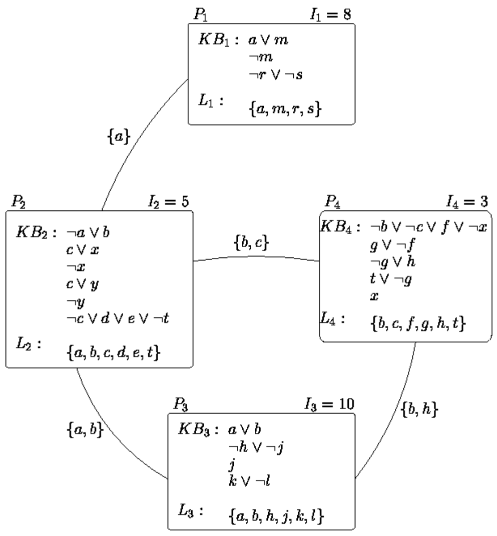

Figure 5: A $\mathcal{P}$-inconsistent PQAS

In this setting, we will again be interested in the query f to peer P_4. In the $\mathcal{P}$-inconsistent setting, however, we need to ask both the query and its negation.

We start by asking the query itself. One trace of our message-passing algorithm may then look as shown in Figure 6. Although a derivation of the empty clause is found for each split literal of the clause $\neg b \vee \neg c$ at the top, this trace does not constitute a valid proof. Notice that the clause x of KB_4 is used to derive the clause $\neg b \vee \neg c$ and the clause $\neg x$ of KB_2 is used to derive $\Box$ from $\neg c$. Thus both x and $\neg x$ are part of $\Box.OA$ which is therefore unsatisfiable. The following clauses are used in this failed proof attempt, where OA clauses (i.e. all but the negated query) are underlined: $\neg f$, $\underline{\neg b \vee \neg c \vee \neg f \vee \neg x}$, $\underline{x}$, $\underline{\neg a \vee b}$, $\underline{a \vee m}$, $\underline{\neg m}$, $\underline{c \vee x}$, and $\underline{\neg x}$.

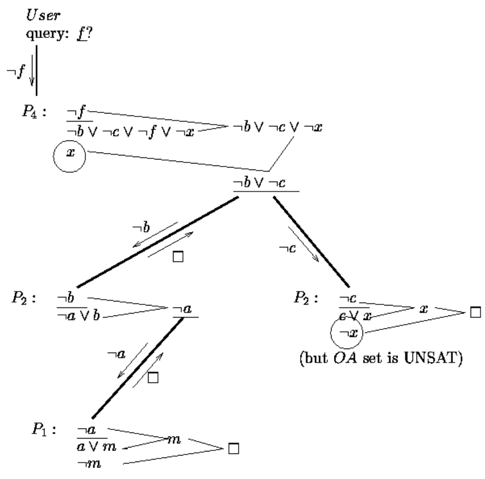

Figure 6: Not a proof of f

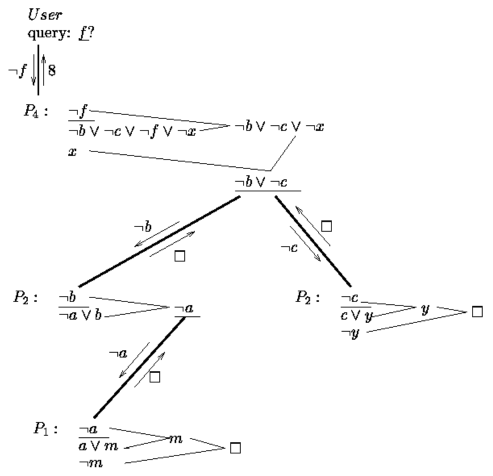

Figure 7: A proof of f

However, the query f is still provable from a consistent subset of the global KB. Figure 7 visualizes a proof (i.e. the derivation of a reason for f). The set of clauses used is the same as above, except that $c \vee x$ and $\neg x$ are replaced by $c \vee y$ and $\neg y$, which avoids the inconsistency hailing from $\neg x$.

Having established a reason of rank 8 for the query f itself, we need to establish whether the negation of the query can be derived in the system as well, and if so, with what rank. Figure 8 shows that $\neg f$ can indeed be proved by the system with rank 10. Since the priority of the reason for $\neg f$ is worse than that of the reason for f, f is to be believed over $\neg f$.

4.2.4 Properties of the algorithms

In this subsection we discuss some soundness, termination, and completeness properties of the algorithms for $\mathcal{P}$-inconsistent PQASs. In particular, we will show that the recursive algorithm is sound and complete with respect

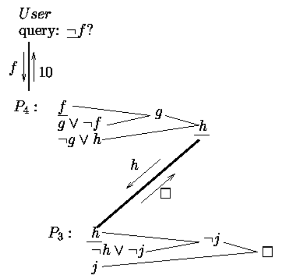

Figure 8: A proof of $\neg f$

to distributed entailment for $\mathcal{P}$-inconsistent PQASs and is guaranteed to terminate. We then show that the message-passing algorithm computes the same results as the recursive one, and therefore enjoys the same properties.

We first need some intermediate results, namely soundness and completeness with respect to computing reasons. The algorithm is considered sound with respect to computing reasons as defined in Definition 3.11 if the OA set of each empty clause it returns represents a reason in support of the query. It is considered complete with respect to computing reasons if for each reason that exists in support of the query, the algorithm finds an empty clause with an OA set corresponding to the reason. The following two theorems demonstrate soundness and completeness of the recursive query-answering algorithm with respect to computing reasons.

Theorem 4.11 (Soundness wrt. computing reasons)**.** *Given a $\mathcal{P}$-inconsistent PQAS $\mathcal{P}$ and a peer P_i, for every empty clause $\Box$ that Find_Bottom($\neg q, \{P_i\}, \emptyset$) of Algorithm 7 returns, there exists a set of formulas Σ s.t. $\mathcal{P} \models_i^\Sigma q$.*

Proof. P may be a peer in $\mathcal{P}$ or an additional query peer that is connected to peers in $\mathcal{P}$. Find_Bottom($\neg q, \{P_i\}, \emptyset$) is the top-level call for the general case of Find_Bottom($p, SP, hist$), where the history has the form $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1)]$ for $n \geq 0$. The h_m are the literals that the derivation of p in the current reasoning branch depends on. We will show that for each $\Box$ that Find_Bottom($p, SP, hist$) returns, a set of formulas Σ and a peer $P_i \in SP$ exist s.t. $\mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p$. We proceed by induction on the number of recursive calls the algorithm requires to terminate.

Base case (no recursive calls). Find_Bottom($p, SP, hist$) returns empty clauses for all $\Box \in$ RESULT. **(i)** On line 11 of Algorithm 7, $\Box$ is added to the set RESULT if $(\neg p, \_, \_) \in hist$ and for the last clause added to the history (c' if the history has the form $[(l', \_, c')|hist']$), $c'.OA$ is satisfiable. Clearly, because $\neg p \to \neg p$, we have $\mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p$ since one of the $h_m = \neg p$. $\Sigma = c'.OA$ as the OA set (without the negation of the original query) of the clause, of which $\neg p$ is a literal, can be viewed as its reason. $\Sigma \in KB_i$ since this run of the algorithm requires no recursive calls. Since $c'.OA$ is satisfiable, $\Sigma \not\models \bot$. This is condition (1) of Definition 3.11 with $\phi = \bigwedge_{m=1}^n h_m \to \neg p$ and therefore $\mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p$. **(ii)** On line 17, $\Box$ is added to RESULT as a local consequence in P_i. Since the empty clause is locally derived from p in P_i given the literals h_m from the history, we have

$$\begin{aligned} & KB_i \cup \{p\} \models_i \bigwedge_{m=1}^n h_m \to \bot \\ \Rightarrow\ & KB_i \models_i \bigwedge_{m=1}^n h_m \to \neg p \\ \Rightarrow\ & \mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p \end{aligned}$$

where $\Sigma = \Box.OA$ since $\Box.OA$ collects all original ancestors of the empty clause except the negation of the original query. By the condition on line 17 of the algorithm, $\Box.OA$ is satisfiable and therefore $\Sigma \not\models \bot$. $\Sigma \in KB_i$ since this run of the algorithm requires no recursive calls. Again this is condition (1) of Definition 3.11 with $\phi = \bigwedge_{m=1}^n h_m \to \neg p$ and therefore $\mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p$.

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and if Find_Bottom$(l, SP', [(p, P_i, _)|hist])$ returns an empty clause $\Box$ then there is a peer $P_j \in SP'$ and a set of formulas Σ_j s.t. $\mathcal{P} \models_j^{\Sigma_j} \bigwedge_{m=1}^n h_m \wedge p \to \neg l$. Let Find_Bottom$(p, SP, hist)$ be a call to the algorithm that requires r recursive calls to terminate.

In this case, an empty clause is returned if there exists a clause c s.t. $KB_i \cup \{p\} \models c$ and for all literals $l \in c$, Find_Bottom$(l, ACQ(l, P_i), [(p, P_i, _)|hist])$ returns an empty clause $\Box$. By the induction hypothesis, we have for each $l \in c$ that for some peer $P_j \in ACQ(l, P_i)$, $\mathcal{P} \models_j^{\Sigma_l} \bigwedge_{m=1}^n h_m \wedge p \to \neg l$ where Σ_l is satisfiable **(I)**. By Definition 4.3 of ACQ, $P_j \in ACQ(l, P_i)$ implies that $(i, j, L_{ij}) \in E$ where $l \in L_{ij}$ **(II)**. Consider now the extension of the local knowledge base KB_i by the set of clauses $\{\bigwedge_{m=1}^n \neg h_m \vee \neg p \vee \neg l\}_{l \in c}$. Then we have

$$
\begin{aligned}
& KB_i \cup \{ \bigwedge_{m=1}^{n} \neg h_m \vee \neg p \vee \neg l \}_{l \in c} \models_i \bigwedge_{m=1}^{n} \neg h_m \vee \neg p \\
\Rightarrow \quad & KB_i \cup \{ \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l \}_{l \in c} \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \neg p \;\; \textbf{(III)}
\end{aligned}
$$

where the entailment in the first line is true since $KB_i \models_i c$ and c resolves with the $\{\neg l\}_{l \in c}$ under the classical entailment relation $Cons_i$. The second line is obtained from the first one by rewriting the disjunctions on either side of the $\models_i$ as implications. Let $\Sigma' = c.OA$. Then $\Sigma = \Sigma' \cup \bigcup_{l \in c} \Sigma_l$ and $\Sigma \not\models \bot$ as $\Box.OA$ is satisfiable if $\Box$ is returned by the condition on line 31 **(IV)**.

But **(I)**, **(II)**, **(III)**, and **(IV)** are just condition (2) of Definition 3.11 with the $\psi_j = \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l$ for all $l \in c$ and $\phi = \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$. Therefore, by the definition of support of a formula by a reason, $\mathcal{P} \models_i^{\Sigma} \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$ and we are done. ■

Theorem 4.12 (Completeness wrt. computing reasons)**.** *Given a $\mathcal{P}$-inconsistent PQAS $\mathcal{P}$ and a peer P_i, if $\mathcal{P} \models_i^{\Sigma} q$ for some set of formulas Σ, then Find_Bottom($\neg q, \{P_i\}, \emptyset$) of Algorithm 7 returns an empty clause $\Box$ s.t. $\Box.OA = \Sigma$.*

Proof. P may be a peer in $\mathcal{P}$ or an additional query peer that is connected to peers in $\mathcal{P}$. Find_Bottom($\neg q, \{P_i\}, \emptyset$) is the top-level call for the general case of Find_Bottom($p, SP, hist$), where the history has the form $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1)]$ for $n \geq 0$. The h_m are the literals that the derivation of p in the current reasoning branch depends on. We will show that if for some peer P_i and some set of formulas Σ, $\mathcal{P} \models_i^{\Sigma} \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$, then $\Box$ with $\Box.OA = \Sigma$ is returned by Find_Bottom($p, SP, hist$). We do so by induction on the number of recursive applications of Definition 3.6 in $\mathcal{P} \models_i^{\Sigma} \bigwedge_{m=1}^{n} h_m \rightarrow \neg p$.

Base case (no recursive applications). In this case we have by condition (1) of the definition

$$\begin{aligned} & \mathcal{P} \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p \\ \Rightarrow\ & \Sigma \models_i \bigwedge_{m=1}^n h_m \to \neg p \\ \Rightarrow\ & \Sigma \cup \{\neg p\} \models_i \bigwedge_{m=1}^n h_m \to \bot \end{aligned}$$

for some $\Sigma \in KB_i$. If one of the $h_m = \neg p$, i.e. the above is trivially true, then $(\neg p, \_, \_) \in hist$ (line 5 of Algorithm 7) and $\Box$ is returned with $\Box.OA = \Sigma$ since $\Box.OA$ is the set of formulas that derives $\neg p$ ($\bot$ is derived from $\Sigma \cup \{p\}$). hier

Otherwise, line 12 computes the set $\{\phi | \Sigma^* \cup \{\neg q\} \models_i \bigwedge_{m=1}^n h_m \to \phi$ for some $\Sigma^* \in KB_i\}$. By the assumption, the empty clause $\Box$ with reason Σ is in this set and if $\Sigma \not\models \bot$ then $\Box.OA$ is satisfiable. Therefore the algorithm returns $\Box$ with $\Box.OA = \Sigma$ by the condition on line 17.

Induction step (r recursive applications). Assume that for $r' < r$ recursive applications of the definition the induction hypothesis is true and if there is a peer $P_j \in SP'$ and a set of formulas Σ s.t. $\mathcal{P} \models_j^{\Sigma_j} \bigwedge_{m=1}^n h_m \wedge p \to \neg l$ then Find_Bottom$(l, SP', [(p, P_i, _)|hist])$ returns $\Box$ with $\Box.OA = \Sigma_j^*$. $\Sigma_j^* \supseteq \Sigma_j$ since $\Box$ is derived by the algorithm using l (derived from Σ_j) and $\neg l$ (derived from some additional clauses from the peer which derived $\neg l$). Let $KB_i \models_i^\Sigma \bigwedge_{m=1}^n h_m \to \neg p$ require r recursive applications of the definition of $\models_i^\Sigma$.

Since at least one recursive application of the definition is required, condition (2) applies and we have

$$\begin{aligned} & \Sigma' \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^n h_m \to \neg p \textit{ with } \Sigma' \subseteq KB_i \ \ (\mathbf{I}) \textit{ and} \\ & \textit{for each } \psi_j\ \exists P_k, \Sigma_j \textit{ s.t. } \mathcal{P} \models_k^{\Sigma_j} \bigwedge_{m=1}^n h_m \to \psi_j \ \ (\mathbf{II}) \textit{ and} \\ & (i, k, L_{ik}) \in E \textit{ where } L_{ik} \supseteq sig(\psi_j) \ \ (\mathbf{III}) \textit{ and} \\ & \Sigma \not\models \bot \textit{ for } \Sigma = \Sigma' \cup \bigcup_j \Sigma_j \ \ (\mathbf{IV}). \end{aligned}$$

(I) can be rewritten as

$$\Sigma' \cup \{p\} \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \bot.$$

By refutation completeness and since Σ' is consistent, there must exist a clause c s.t.

$$\begin{aligned} &\Sigma' \cup \{p\} \models_i c \\ \textit{and} \quad &\{c\} \cup \{\psi_j\}_j \models_i \bigwedge_{m=1}^{n} h_m \rightarrow \bot. \end{aligned}$$

Since the ψ_j are derived from the Σ_j and $\bigcup_j \Sigma_j$ is consistent by **(IV)**, the set $\{\psi_j\}_j$ is consistent and does not entail $\bot$. So $\bot$ must result from clauses in $\{\psi_j\}_j$ resolving with all literals $l \in c$. c is a descendant of the negation of the original query and as such must be involved in deriving $\bot$ from an otherwise consistent set. The $\{\psi_j\}_j$ must thus entail $\neg l$ for each $l \in c$. Without loss of generality, we can thus assume that **(I)** holds and can be rewritten in the form

$$\Sigma' \cup \{\neg l\}_{l \in c} \models_i \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \bot.$$

Furthermore, **(II)** is of the form

$$\mathcal{P} \models_k^{\Sigma_l} \bigwedge_{m=1}^{n} h_m \wedge p \rightarrow \neg l$$

for some peer P_k for each $l \in c$. But for this case,
Find_Bottom$(l, ACQ(l, P_i), [(p, P_i, _)|hist])$ of line 23 of Algorithm 7 returns $\Box_l$ with satisfiable $\Box_l.OA$ for each $l \in c$ by the induction hypothesis and since $P_k \in ACQ(l, P_i)$ because $(i, k, L_{ik}) \in E$ with $l \in L_{ij}$ by **(III)**. We can invoke Lemma 4.18 to realize that for each $l \in c$, the empty clause $\Box_l$ that corresponds to the reason Σ_j is returned as part of the set ANSWER(l) on line 23.

Since for each $l \in c$, the empty clause with OA set corresponding to the reason Σ_j is returned by the recursive call, the disjunction of these empty clauses will be an empty clause with OA set $\Box.OA = \bigcup_j \Box_j.OA$. Since $\Sigma = \Sigma' \cup \bigcup_j \Sigma_j$ is consistent by **(IV)** and each $\Box_l.OA$ corresponds to its Σ_j, $\Box.OA = \Sigma \cup \{p\}$ is satisfiable and the algorithm Find_Bottom$(p, SP, hist)$ returns $\Box$ with OA set $\Box.OA$ on line 32 and we are done. ■

Lemma 4.13 (Merging empty clause consequences)**.** *Let $c = l_1 \vee \cdots \vee l_m$ be a clause. For every way to derive the empty clause $\Box$ from c, there exist $\Box_1, \ldots, \Box_m$ such that $\Box = \Box_1 \vee \cdots \vee \Box_m$ and for every i, $\Box_i$ is a consequence of l_i*

Proof. In order to derive the empty clause $\Box$ from c by resolution, every literal of c has to be "resolved away". The set C of clauses that resolve literal l_i of c away must therefore entail $\neg l_i$. Since $C \models \neg l_i$, $C \cup \{l_i\} \models \bot$. Thus an empty clause $\Box_i$ is a consequence of each literal l_i of c. ■

We have shown that Algorithm 7 is sound and complete with respect to computing reasons. That is, Algorithm 7 returns $\Box$ iff the query is supported by the reason corresponding to $\Box.OA$. The following theorem verifies that Algorithm 7 is guaranteed to terminate, i.e. the user will eventually receive an answer for the query.

Theorem 4.14 (Termination (following [4]))**.** *Algorithm 7 terminates.*

Proof. The proof is the same as that of Theorem 4.8 on termination of the recursive query answering algorithm for $\mathcal{P}$-consistent PQASs. ■

We now have the tools to show soundness and completeness of the recursive Algorithm 6 with respect to distributed entailment for $\mathcal{P}$-inconsistent PQASs as described in Definition 3.12. The algorithm is considered sound with respect to distributed entailment if whenever it returns YES, the original query is supported by a reason and all reasons in support of the negated query are of lower rank. It is considered complete with respect to distributed entailment if it returns YES for every query for which a reason exists and for which all reasons for the negation are of lower rank.

Theorem 4.15 (Soundness and completeness wrt. distributed entailment)**.** *Given a $\mathcal{P}$-inconsistent PQAS $\mathcal{P}$ and a peer P, Answer_Query(q, P) of Algorithm 6 returns YES iff $\mathcal{P} \models_D q$.*

Proof. $\Rightarrow$: Assume that Answer_Query(q, P) returns YES. We will show that $\mathcal{P} \models_D q$. Answer_Query(q, P) returns YES iff the set Pos contains a higher-rank reason than the set Neg. By the soundness, completeness, and termination of Algorithm 7 (Theorems 4.11, 4.12, and 4.14), Pos of line 2 contains exactly all reasons for q and Neg of line 3 contains exactly all reasons for $\neg q$. So Answer_Query(q, P) returns YES iff there exists a reason supporting q and all reasons supporting $\neg q$ are of lower rank (line 12). But this is exactly the Definition 3.12 of distributed entailment and thus $\mathcal{P} \models_D q$.

$\Leftarrow$: Assume that $\mathcal{P} \models_D q$. We will show that Answer_Query(q, P) returns YES. By Theorem 4.14, Find_Bottom terminates and lines 2 and 3 of Algorithm 6 will return two sets Pos and Neg with reasons for both q and $\neg q$, respectively. Since Find_Bottom is sound and complete for computing reasons by Theorems 4.11 and 4.12, Pos and Neg will contain exactly *all* the reasons for q and $\neg q$. Given that the highest-rank reason for each is selected on lines 7 and 11, YES will be returned on line 13 iff the positive query q is supported by a higher-rank reason than the negation of the query $\neg q$ (line 12), which is the case by the assumption that $\mathcal{P} \models_D q$. ■

Theorems 4.15 and 4.14 show that the recursive algorithm for the $\mathcal{P}$-inconsistent case has similar properties as that for the $\mathcal{P}$-consistent one, namely soundness and completeness with respect to distributed entailment as well as termination. In order to show that our message-passing algorithm for $\mathcal{P}$-inconsistent systems also benefits from these properties with respect to computing reasons, we show the correspondence between the recursive and message-passing algorithms in the following theorem. In Theorem 4.16 we ignore the priority check of clauses and peers in Algorithm 8 (lines 19 and 32) and Algorithm 9 (line 9) to show that without such pruning the recursive and message-passing algorithms compute the same results. We will show later in Theorem 4.17 that the pruning technique only results in non-optimal rank reasons not being found, which is not a restriction (but a time saving) as we are interested in the best-rank reasons for the query and its negation only. The following theorem and its proof are an adaptation of Theorem 4.9 to this far more complex $\mathcal{P}$-inconsistent case.

Theorem 4.16 (Correspondence between recursive and message-passing algorithms)**.** *For each empty clause $\square$ returned by Find_Bottom($p, SP, hist$) of Algorithm 7, if a peer P sends the message $m(P, P', forth, hist, p)$ (Algorithm 8, without pruning) to some peer $P' \in SP$, it receives the message $m(P', P, back, [(p, P', \square)|hist], \square)$ (Algorithm 9, without pruning). Furthermore, P will eventually receive a message $m(P', P, final, hist)$ (Algorithm 10).*

Proof. **Equivalence of algorithms' results.**

Note again that we assume here that no clause or peer is ignored (pruned) for having worse priority than the value of *best*.

We will first prove by induction on the number of recursive calls of Algorithm 7 the equivalence of results by showing that a peer P receives a message $m(P', P, back, [(p, P', \square)|hist], \square)$ from some peer $P' \in SP$ in the system for each $\square$ that Find_Bottom$(p, SP, hist)$ returns.

Base case (no recursive calls). Find_Bottom$(p, SP, hist)$ returns $\square$ if $(\neg p, \_, \_) \in [(l', \_, c')|hist']$ and $c'.OA$ is satisfiable (line 5) or an empty clause $\square$ with satisfiable OA set is a local consequence (line 17). The former case corresponds to line 7 in Algorithm 8, in which case the message $m(P', P, back, [(p, P', \square)|hist], \square)$ is sent (line 13). The latter case corresponds to line 23 of Algorithm 8, in which case the *back* message is sent as well.

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and for all empty clauses (with satisfiable OA set) returned by calls of the recursive algorithm a corresponding *back* message is sent by the message-passing algorithm. Let Find_Bottom$(p, SP, hist)$ be a call to the algorithm that requires r recursive calls to terminate.

Find_Bottom$(p, SP, hist)$ returns $\square$ if for each literal of some local consequence, the algorithm recursively returns $\square$ and the combined OA set is satisfiable (line 31). Let $c = p_1 \vee \cdots \vee p_m$ be a local consequence of a peer P' such that for each p_i, a recursive call Find_Bottom$(p_i, ACQ(p_i, P'), [(p, P, c)|hist])$ returns at least one $\square_i$ from a peer $P_i \in ACQ(p_i, P')$. By the induction hypothesis, a message $m(P_i, P, back, [(p_i, P_i, \square_i)|hist_i], \square_i)$ is sent for each i. For each such message, Algorithm 9 records that literal p_i generated the *back* message (line 3). When a *back* message for one of c's literals is received and a new empty clause for the original consequence c results (line 5), a *back* message $m(P', P, back, [(p, P', \square)|hist'], \square)$ is sent since c was a local consequence of p in P' (line 16 of Algorithm 9). P' is the peer having sent a *forth* message to P to request empty clause consequences of p.

Notification of termination. It remains to be shown that the message-passing algorithm notifies the user peer of termination once it occurs. In other words, we need to show that once all possible derivations of the empty clause as a consequence of a literal have been found, or it has been shown that the empty clause is not a consequence, a *final* message $m(P', P, final, hist)$ is sent. Again we will proceed by induction.

Base case (no recursive calls). The empty clause is not a consequence of p in P' if either $(p, \_, \_)$ already occurs in the history (i.e. we have a cycle and all potential consequences have been found on the first visit already) or no consequences with all shared literals are locally derived. In both cases a *final* message is sent (lines 3 and 30 of Algorithm 8). These are the only two possibilities for the case of no recursive calls since local consequences with all shared literals would trigger a recursive call of Algorithm 7 (a *forth* message in Algorithm 8).

Induction step (r recursive calls). Assume that for $r' < r$ recursive calls the induction hypothesis is true and the top-level call of the algorithm receives *final* messages for each of its recursive calls eventually. Let Find_Bottom($p, SP, hist$) be a call to the algorithm that requires r recursive calls to terminate.

A *final* message needs to be sent once a *back* message is received for each literal in a local consequence with all shared literals as well as when a *final* message has been received for each literal in such a local consequence. Line 9 of Algorithm 10 ensures that this is the case. ■

Due to Theorem 4.16, the soundness and completeness properties with respect to computing reasons of Theorems 4.11 and 4.12 carry over to the message-passing version of the algorithm. Furthermore, the user is again eventually notified of its termination. Termination in the $\mathcal{P}$-inconsistent case occurs when the search for reasons space is exhausted, whether derivations of the empty clause are found or not (since we search for the *best* reason in terms of priority). However, the exhaustion of the search space is accelerated by the pruning technique of keeping track of the rank of the best currently known reason and ignoring all consequences and peers with worse priority. The *prio* messages exchanged between peers when a reason is found spread the "news" about a new current best-rank reason being known to the query peer.

Theorem 4.17 below verifies that the algorithm still finds the best-rank reason for either the query or its negation and thus ensures the correct computation of whether the query or its negation is entailed by distributed entailment.

Theorem 4.17 (Pruning by priority limit)**.** *The following hold for a PQAS running the message-passing procedures of Algorithms 8, 9, 10, and 11. (i) Whenever a peer updates its local value of best, there exists a reason for the original query or its negation with priority best. (ii) Local consequences and acquainted peers ignored by a peer P cannot result in a reason for the original query or its negation with better or equal priority than the current value of best.*

Proof. **(i)** A *prio* message is sent iff a new reason for the original query or its negation of better priority than the current local value of *best* has been found. A reason for the original query or its negation can only be found in the query peer as either a top-level call or a complete unwinding of the virtual "call stack" represented by the *forth* and *back* messages. This is the case iff the history is empty. A new reason is thus found iff an empty clause has been derived and the current message's history is empty. This is the case on lines 14 and 26 of Algorithm 8 and line 18 of Algorithm 9. A peer only updates its local value of *best* when it receives a corresponding *prio* message (line 3 of Algorithm 11).

(ii) A peer P ignores clauses of priority worse than *best* on line 21 of Algorithm 8 and line 9 of Algorithm 9. It furthermore ignores acquainted peers of priority worse than *best* on line 34 of Algorithm 8. It ignores a $forth$ message if either the corresponding literal stems from a clause of worse priority than *best* or the priority of the receiving peer is worse than *best* (Algorithm 8, line 4). Ignoring a peer of priority I amounts to ignoring clauses of the same priority as each clause in a peer has the priority of the peer. Thus is suffices to show that no reason (empty clause consequence) of priority better than *best* can be found by resolving with a clause c of priority $c.prio > best$.

Upon resolving with a clause c of priority $c.prio > best$, a new element with the resolvent c' is added to the history. By lines 20 of Algorithm 8 and 9 of Algorithm 9, we have $c'.prio \geq c.prio > best$. By the monotonicity of priorities in a history progression (Lemma 4.18), no consequence of c' (including a potential empty clause found) will have priority better than $c'.prio > best$.

In case the processing of a *forth* generates no empty clauses and no further *forth* messages (e.g. because all neighboring peers have worse priority than *best* and are thus pruned), a *final* message needs to be sent on line 39 of Algorithm 8. ■

Lemma 4.18 (Monotonicity of priorities)**.** *For a history of the form* $[(l_n, P_n, c_n), \ldots, (l_i, P_i, c_i), \ldots, (l_1, P_1, c_1), (\neg q, P_0, c_0)]$*, the values of* $c_i.prio$ *are monotonically increasing with increasing* i*. I.e. for* $j > i$*,* $c_j.prio \geq c_i.prio$*.*

Proof. An element gets added to the front of the history if a new consequence is derived. The priorities of new consequences are set on lines 12 and 20 of Algorithm 8 and line 8 of Algorithm 9. On line 12 of Algorithm 8, $\Box.prio$ is set to the priority of the clause in the last element previously added to the history, so $\Box.prio = c'.prio$.

On line 20 of Algorithm 8, a newly found consequence c's priority is set to the maximum of all clauses c_i^*'s priorities that went into the new consequence. Hence, $c.prio \geq c_i^*.prio$ for all i. In particular, $c.prio \geq p.prio$ as p is one of the c_i^* and since a literal of the clause most recently added to the history has the same priority as that most recently added clause.

On line 8 of Algorithm 9, $\Box.prio$ is set to the maximum of the priorities of the empty clauses $\Box_i^*$ that it is made up of. Each of the $\Box_i^*$ is a local consequence in some peer and has therefore greater or equal priority value as the literal it was locally induced by (as shown above). That literal was part of the most recently added clause c to the history and since the maximum is taken, $\Box.prio \geq c.prio$. ■

4.2.5 Priority-ordering heuristic

In the previous subsection we observed that we need to know only the best-rank reason for either the query or its negation, but not all reasons, in order to answer a query. This allowed us to prune clauses and peers of worse priority than the currently best known rank of a reason. Until now, we only passively recorded the best known reasons in the order they were naturally found. However, we have not tried to actively find the better-rank reasons first. In this subsection we introduce a heuristic that tends to direct the search towards the best-rank reasons, so they will be found earlier and enable earlier pruning of larger sections of the search space.

Let us take a closer look at what happens at the peers when messages are sent and received. The implementation of a PQAS requires a message queue at each peer, since messages may be sent more quickly than they can be processed by a receiving peer, especially since messages are often sent in waves (e.g. a set of *forth* messages for each literal of a consequence clause). By default, this queue is a first-in first-out (FIFO) queue that processes messages in the order in which they arrive. But since the reasons resulting from a clause can never have better rank than the priority of the clause (by Lemma 4.18), it pays off to consider better-priority consequence clauses and acquatined peers first. This observation justifies the use of a *priority* queue as the message queue of each peer. The messages in the priority queue will be ordered by the priorities of the clauses they contain, and the best-priority messages will be processed first. An exception will be *prio* messages, which will receive priority 0 and thus will always be processed first in order to adjust the pruning limit *best* as soon as a new best-rank reason is found. Messages of equal priority will be processed in the order they are received. The intuition clearly is that if better-priority messages are processed first, the best-rank reason for or against the query will be found sooner and the remaining search space can be pruned earlier. The following theorem shows that our algorithm is still sound and complete with respect to finding the best-rank reason for the query or its negation.

Theorem 4.19 (Soundness and completeness wrt. best-rank reasons)**.** *Algorithms 8, 9, 10, and 11 are sound and complete with respect to finding the best-rank reason for the query or its negation when all peers employ a priority message queue.*

Proof. This follows immediately from Theorem 4.17. A peer's local value of *best* is never better than a currently known reason for or against the query and no pruned consequence clauses or acquainted peers can result in a reason of better rank than *best*. Thhus, if the algorithms run to completion, the best-rank reason for the query or its negation will be found, if one exists. ■

We can take advantage of the order of priorities through one more small optimization. While the priority message queues at the peers ensure that among the received messages, the better-priority ones are processed first, there cannot be a global mechanism leading to better-priority messages to be processed before worse-priority ones. However, we can try to send better-priority messages before worse-priority ones. In particular, we can send *forth* messages to better-priority acquaintances first. This requires simply to go through acquainted peers in the order of their priorities on line 35 of Algorithm 8. Sending *forth* messages in order of the receiving peers' priorities allows better-priority messages to arrive at the respective receiving peer earlier and thus to get processed sooner.

In Section 6 we describe an implementation of a PQAS simulation and present experimental results illustrating the effectiveness of the priority-ordering heuristic introduced in this subsection.

4.2.6 Prioritized $\mathcal{P}$-consistent Systems

In Section 4.1, we ignored the priorities of $\mathcal{P}$-consistent PQASs. However, priorities may still carry valuable information in this setting. The priorities may, for example, correspond to the reliability or trustworthiness of a peer. In this case, not only the truth of a query, but also its reliability may be of interest. In particular, the user may be interested in finding the most reliable reason for the query and its reliability.

In Section 3.2 we defined the reason for a formula (Definition 3.8) as well as the rank of a reason (Definition 3.9) for $\mathcal{P}$-consistent PQASs. A reason for a formula in a $\mathcal{P}$-consistent PQAS is defined similarly to a reason in the $\mathcal{P}$-inconsistent case, except that in $\mathcal{P}$-consistent PQASs all formula sets are necessarily consistent and a reason's consistency does not need to be explicitly required. Our algorithm for the $\mathcal{P}$-inconsistent case is already capable of dealing with and aggregating priorities. Algorithm 7 already computes all reasons for a given single-literal query in $\mathcal{P}$-inconsistent PQASs and can thus be used to do the same for $\mathcal{P}$-consistent PQASs. However, in the $\mathcal{P}$-consistent case, OA sets do not need to be kept track of if only the rank of the best-rank reason is of interest since OA sets are necessarily consistent in a $\mathcal{P}$-consistent PQAS. This also saves us the expensive satisfiability checks of OA sets. Likewise, the message-passing version of the algorithm for the $\mathcal{P}$-inconsistent case can be used for finding best-rank reasons in the $\mathcal{P}$-consistent case. Again, the satisfiability of potential reasons need not be checked in $\mathcal{P}$-consistent PQAS. Thus Algorithms 8, 9, 10, and 11 (with OA tracking and satisfiability checking turned off) can be used to find the best-rank reason in $\mathcal{P}$-consistent PQASs.

4.3 Complex Queries

So far in this section we have focused on single-literal queries. Our techniques are, however, also able to deal with more complex queries. This subsection talks about conjunctive and disjunctive queries and which our algorithms can and cannot handle.

4.3.1 Conjunctive queries

The simplest kind of composite queries are conjunctions of literals, as defined straightforwardly below.

Definition 4.20 (Conjunctive query). *A conjunctive query is a conjunction of literals $l_1 \wedge \cdots \wedge l_n$ that succeeds iff each individual literal query succeeds.*

We concentrate here on the recursive version of the algorithm for the $\mathcal{P}$-consistent case; the $\mathcal{P}$-inconsistent case works almost analogously and is discussed briefly below. For a conjunctive query to succeed, each individual literal must be provable in the peer-to-peer reasoning system. Thus the simplest way to solve a conjunctive query is to split it up into its individual literals l_i and to call the appropriate algorithm n times, once for each l_i. The original conjunctive query then succeeds iff the algorithm returns YES for *each* literal l_i. Algorithm 12 summarizes this procedure.

Algorithm 12 Recursive algorithm for conjunctive queries in a $\mathcal{P}$-consistent PQAS

```
1: **Answer_ConjQuery**($\phi, P$)
2: **for all** literals $l_i \in \phi$ **do**
3:     ANSWER$_i \leftarrow$ Adjiman_QA($\neg l_i, \{P\}$)
4: **if** for all ANSWER$_i$, ANSWER$_i = YES$ **then**
5:     **return** $YES$
6: **else**
7:     **return** $NO$
```

In the $\mathcal{P}$-inconsistent case, Find_Bottom of Algorithm 7 needs to be modified to return *all* found reasons $\Box_i$ of a literal l_i, not only the best-rank reason. The conjunctive query answering algorithm would then find all possible combinations of single-literal reasons whose $\Box_i.OA$ sets are mutually satisfiable. The set of reasons for the original conjunctive query would then be $Reasons \leftarrow \{\Box \in \otimes_{i=1..n}\Box_i | \Box.OA \text{ is SAT}\}$. The rank reported is the best rank $\max_{\Box \in Reasons}(\Box.prio)$.

In the $\mathcal{P}$-inconsistent case, of course, the negation of the query needs to be handled as well (to compare ranks of reasons for the query and its negation). The negation of a conjunctive query is a disjunction, and disjunctive queries are difficult to deal with as we will see in the next subsection.

An easier way to achieve conjunctive query answering is to negate the entire conjunctive query first and then handle the resulting disjunction like any other clause in the query answering algorithms discussed in this section. In other words, the disjunction resulting from the negation of the original query is split into its individual literals, the algorithms recursively called (or a set of *forth* messages) sent for each literal, and the returned results combined using the distributed disjunction operator $\otimes$. If the empty clause (with a satisfiable OA set in the $\mathcal{P}$-inconsistent case) is a consequence of the negated conjunctive query (a disjunction, i.e. clause), then the original query is entailed by the system.

4.3.2 Disjunctive queries

Disjunctive queries turn out to be much more difficult to handle and cannot be decomposed and pieced back together in the general case. However, this strategy does work in a special case discussed below.

Definition 4.21 (Disjunctive query)**.** *A disjunctive query is a disjunction of literals* $l_1 \vee \cdots \vee l_n$.

Let us start by examining more closely when a disjunctive query should succeed. In general, a disjunction queries must succeed iff the disjunction is entailed by the system. Our algorithms cannot compute directly whether a disjunction is entailed by the system. However, as a special case, a disjunction $l_1 \vee \cdots \vee l_n$ clearly is entailed if *at least one* of the individual literals l_i is entailed individually:

$$\exists i.KB \models l_i \Rightarrow KB \models \bigvee_i l_i$$

for some knowledge base KB, such as the global KB of a PQAS. In this special case, a decomposition technique for the disjunctive query clearly works. We split up the query into its individual literals l_i again and the original query succeeds if at least one of the l_i queries succeeds. This procedure, shown in Algorithm 13, is again for the $\mathcal{P}$-consistent case. In the $\mathcal{P}$-inconsistent case, the best-rank reason for any of the literals is picked as the best-rank reason for the entire (disjunctive) query. The satisfiability of the OA set for that literal's reason alone is sufficient to constitute a reason for the whole query since only one literal of the disjunctive query needs to be shown true for the whole query to be true.

In the general case, however, the literals of a disjunctive query are not always entailed individually. Consider for example a scenario in which a system entails $s \vee t$ but does not entail either s or t alone. Then the decomposition approach to processing a disjunctive query will yield a NO answer while, in fact, the query $s \vee t$ is entailed. Our refutation proof approach of recursively generating and returning empty clauses on a per-literal basis will thus not work here. It would be an interesting area of future work to modify or extend the algorithms discussed in this section to handle all disjunctive queries.

Algorithm 13 Incomplete recursive algorithm for disjunctive queries in a $\mathcal{P}$-consistent PQAS

```
1: **Answer_DisjQuery**($\phi, P$)
2: **for all** literals $l_i \in \phi$ **do**
3:     ANSWER$_i \leftarrow$ Adjiman_QA($\neg l_i, \{P\}$)
4: **if** there exists $i$ s.t. ANSWER$_i = YES$ **then**
5:     **return** $YES$
6: **else**
7:     **return** $NO$
```

4.3.3 Database-only systems

While the previous two subsections talked about the hardness of answering complex queries in the general setting, answering arbitrary queries becomes trivial when all peers of a PQAS are databases. What made complex query answering hard in the general PQAS setting is disjunctive information that is entailed as a disjunction without the disjuncts being entailed individually. There is no disjunctive information in databases since a database lists only atomic information. I.e. each “clause” in a database consists of only a single literal. This fact eliminates our problem with disjunctive queries for PQASs all of whose local peer KBs are databases. In such systems, we can answer arbitrary queries by querying the databases for each individual literal of the query and piecing the individual answers back together to form the answer to the original query. Such an arbitrary query can be expressed in conjunctive normal form (CNF). An algorithm to answer a CNF query would look similar to Algorithm 14. The algorithm returns the rank of the best-rank reason for the query and infinity of no reason exists. In its attempt to find the best-rank reason for the query, for each literal, the algorithm queries the relevant database peers in order of decreasing priority and stops once a positive answer has been found. If none of the peers can answer the query with YES, the literal is believed to be false by a global CWA. One could also argue for a semantics according to which only the best-priority database peer for a given literal would be queried, and its answer believed no matter if it is positive or relies on the peer’s local CWA. A clause is then considered true iff at least one of its literals is true. The whole CNF query is considered true iff all of its clauses are true.

Algorithm 14 Answering CNF queries in database PQASs

```
1: let $\mathcal{Q}$ be the query in CNF
2: $rank \leftarrow \infty$
3: for all clauses $c \in \mathcal{Q}$ do
4:    $this\_clause \leftarrow false$
5:    for all literals $l \in c$ do
6:       for all peers $P_i$ that mention $l$, in order of decreasing priority do
7:          if Ask_Query($P_i, l$)= $true$ then
8:             $this\_clause \leftarrow true$
9:             $rank \leftarrow max(rank, I_i)$
10:            break
11:   if $this\_clause = false$ then
12:      return $\infty$
13: return $rank$
```

5 Global Closed-world Reasoning in PQASs

While peers may have conflicting explicit knowledge in their respective knowledge bases, another source of inconsistencies may be peers employing a local closed-world assumption. Such peers would produce negative literals if the corresponding atom is not locally known positively, even if other peers' local KBs do entail the positive literal.

In this section, we formally characterize one possible semantics of global generalized closed-world entailment and show how query answering according to it can be achieved in a restricted class of PQASs. We furthermore illustrate by two examples that other characterizations of global closed-world entailment may be desirable in different situations and suggest informally how query answering in two of these characterizations may be achieved using our algorithms from the previous section. Lastly, we dicuss such different characterization more generally.

5.1 GCWA Entailment from the Global Theory

Let us first consider the case of a PQAS $\mathcal{P}$ whose graph allows all consequences to propagate between peers and whose global theory $\bigcup_{i=1}^{n} KB_i$ is consistent under classical entailment. Furthermore assume that all peers' local reasoners locally make the closed-world assumption (CWA) with respect to their local theories KB_i. The definition of closed-world entailment, due to Reiter [35], is given below.

Definition 5.1 (Closed-world entailment ($\models_{CW}$) [35]). *Let KB be a consistent propositional theory. Let $KB^+ = KB \cup \{\neg p | p \text{ is atomic and } KB \not\models p\}$. KB entails a formula ϕ by closed-world entailment, denoted $KB \models_{CW} \phi$, iff $KB^+ \models \phi$.*

The goal of this subsection is now to achieve globally consistent reasoning in a PQAS all of whose peers locally employ the CWA. Note that in a PQAS with a consistent global KB, one local reasoner may produce a negative literal $\neg q$ by making the closed-world assumption locally while another local reasoner may be able to prove q positively at the same time. If both consequences now propagate through the system, an artificial contradiction arises. Inconsistent closed-world entailments may also come from positive minimal clauses (or prime implicates) that contain two or more literals. In a knowledge base that only contains the clause $p \vee q$, for example, both $\neg p$ and

$\neg q$ would be entailed by CWA entailment and result in an inconsistent KB^+ (following Definition 5.1). We would like to detect when a local closed-world consequence conflicts with an explicitly entailed consequence or a positive prime implicate entailed by the global theory. In other words, we would like a PQAS with a consistent global KB to entail a consistent set of consequences under some kind of global closed-world reasoning.

In order to avoid problems with disjunctions of two or more positive literals in centralized KBs, Minker came up with the generalized closed-world assumption (GCWA), which is equivalent to the regular CWA in theories of only Horn clauses [34]. A Horn clause is defined as containing at most one positive literal. The generalized closed-world assumption, whose definition is stated below, avoids inconsistencies due to minimal positive disjunctions whose literals are not individually entailed classically, by not entailing their negations.

Definition 5.2 (Generalized closed-world entailment ($\models_{GC}$)). *Let KB be a consistent propositional theory. Let $KB^* = KB \cup \{\neg p | p$ is atomic and p occurs in no positive prime implicate of $KB\}$. KB entails a formula ϕ by generalized closed-world entailment, denoted $KB \models_{GC} \phi$, iff $KB^* \models \phi$.*

Given this solution to the problem with positive disjunctions under the CWA in the centralized case, we can now formally specify the global query answering behavior that we desire from a PQAS whose peers make local closed-world assumptions. Given that we assume a globally consistent system in which all local consequences are allowed to propagate between peers sharing their language, it makes sense to define *global generalized closed-world entailment* in terms of entailment from the global theory.

Definition 5.3 (Global generalized closed-world entailment ($\models_{GGC}$)). *Let $\mathcal{P}$ be a PQAS in which each peer locally makes the closed-world assumption (CWA) and the global theory is satisfiable. Then $\mathcal{P}$ entails a formula ϕ by the global generalized closed-world assumption, denoted $\mathcal{P} \models_{GGC} \phi$, iff $\bigcup_{i=1}^{n} KB_i \models_{GC} \phi$.*

In this setting we can solve the query answering problem for single-literal queries using our prioritized algorithms from Section 4. We introduce only a slight adjustment concerning the priorities of some local consequences. Whenever a local reasoner produces a negative literal by making the local closed-world assumption, the resulting (single-literal) clause will receive a special priority value $\mathcal{C}$; all other consequences receive the priority of the lowest-priority peer involved in producing them as usual. The priority $\mathcal{C}$ is lower (greater numeric value) than the priority of any peer in the system, but higher (smaller numeric value) than the lowest possible priority represented by infinity, which is already used to indicate that no reason for a particular query exists. Applying Algorithm 6 in this environment now has the following effect. Asking a negative query which is not entailed classically from the global theory results in a YES answer iff it is entailed by GCWA entailment from the global theory. In particular, negative literals produced as local CWA consequences in one peer are overridden by any positive consequences in other peers since the priority $\mathcal{C}$ of a CWA consequence is always lower than that of a classical consequence. In this setting, we consider the algorithm sound and complete with respect to query answering according to the global generalized closed-world assumption iff the algorithm returns YES for every query entailed by global generalized closed-world entailment, NO for every query whose negation is entailed by global generalized closed-world entailment, and UNK otherwise. According to the GCWA, individual literals' truth values are unknown iff they occur in a positive prime implicate of the theory and are thus neither entailed positively nor negatively by classical entailment. Note that it is enough to require the CWA locally at the peers, not the GCWA, in order to achieve global reasoning in terms of the GCWA. The reason for this is that any queries that should be answered with UNK according to the GCWA are derived both positively and negatively with the same priority $\mathcal{C}$ by our algorithm and are thus, in fact, answered as UNK. Those positive literals occuring in positive prime implicates of the global theory (which are not classically entailed individually) are derived negatively with priority $\mathcal{C}$ by some peers' local CWA. The positive literal itself is also derived with priority $\mathcal{C}$ because all other positive literals of the prime implicate are resolved away by their negative counterparts that are again local CWA consequences. Theorem 5.4 shows that under our assumptions, Algorithm 6 is sound and complete with respect to query answering according to the global generalized closed-world assumption.

Theorem 5.4 (Global generalized closed-world query answering from arbitrary consistent theories)**.** *Let $\mathcal{P} = (P, G)$ be a PQAS with a satisfiable global knowledge base $\bigcup_{i=1}^{n} KB_i$ such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and $Cons_i$ is the CWA entailment relation $\models_{CW}$ s.t. CWA-consequences are assigned the priority $\mathcal{C}$, which is* *lower than any peer's priority but higher than the priority represented by infinity. For a literal q, Answer_Query(q, P) of Algorithm 6 returns YES iff $\mathcal{P} \models_{GGC} q$, NO iff $\mathcal{P} \models_{GGC} \neg q$, and UNK otherwise.*

Proof. By Definition 5.2, $\mathcal{P} \models_{GGC} q$ iff $\bigcup_{i=1}^{n} KB_i \models_{GC} q$. We will thus show that Answer_Query(q, P) returns YES iff $\bigcup_{i=1}^{n} KB_i \models_{GC} q$, NO iff $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$, and UNK otherwise. We will furthermore assume for this proof that the query q is a positive literal. As we will discuss both the query and its negation as well as the cases of the algorithm returning YES and NO, the arguments for a negative-literal query are analogous with the cases for YES and NO being reversed.

YES($\Rightarrow$): Assume that Answer_Query(q, P) returns YES. We will show that $\bigcup_{i=1}^{n} KB_i \models_{GC} q$. Since Answer_Query$(q, P)$ returns YES, q is shown with better rank than $\neg q$. If the rank of the best reason for q is better than $\mathcal{C}$, q is classically entailed by the global theory and thus also $\bigcup_{i=1}^{n} KB_i \models_{GC} q$. In this case, $\neg q$ is not entailed by the global theory as it is assumed to be consistent. If the rank of the best reason for q is $\mathcal{C}$, this means that it involves a positive prime implicate clause for which all literals were resolved away by their negative counterparts produced by the local CWA of individual peers. In this case, however, $\neg q$ is produced in the same way with the same rank $\mathcal{C}$ and thus YES is not returned by the algorithm. This is the required behavior as the negations of positive literals involved in positive prime implicate clauses are not entailed by the definition of the GCWA.

YES($\Leftarrow$): Assume that $\bigcup_{i=1}^{n} KB_i \models_{GC} q$. We will show that Answer_Query(q, P) returns YES. If q is classically entailed by the global theory, the algorithm will find its best-rank reason with rank better than $\mathcal{C}$ (since it is sound and complete wrt. computing reasons). $\neg q$ cannot shown with rank better than $\mathcal{C}$ at the same time since it is not classically because the global theory is assumed to be consistent. Thus q is shown with better rank and YES is returned. If q is not classically entailed by the global theory, it is not entailed by the GCWA at all since it is a positive literal (by assumption).

NO($\Rightarrow$): Assume that Answer_Query(q, P) returns NO. We need to show that $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$. Since Answer_Query$(q, P)$ returns NO, $\neg q$ is shown with better rank than q. If the rank of the best reason for $\neg q$ is better than $\mathcal{C}$, $\neg q$ is classically entailed by the global theory and thus also $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$. In this case, q is not entailed by the global theory as it is assumed to be consistent. If the rank of the best reason for $\neg q$ is $\mathcal{C}$, this means that $\neg q$ is not classically entailed by the global theory and some peer locally produced $\neg q$

by the CWA. Since the algorithm returns NO, the rank for q must be lower (i.e. ∞) and q is not entailed by the global theory. Thus $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$.

NO($\Leftarrow$): Assume that $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$. We need to show that Answer_Query(q, P) returns NO. If $\neg q$ is classically entailed by the global theory, the algorithm will find its best-rank reason with rank better than $\mathcal{C}$ (since it is sound and complete wrt. computing reasons). q cannot be shown with rank better than $\mathcal{C}$ at the same time since it is not entailed classically because the global theory is assumed to be consistent. Thus q is shown with better rank and YES is returned. If $\neg q$ is not classically entailed by the global theory, it is produced locally at some peer by the CWA with rank $\mathcal{C}$. Since $\bigcup_{i=1}^{n} KB_i \models_{GC} \neg q$, q is not entailed classically by the global theory and thus receives rank ∞. It is also not entailed according to the GCWA as it is a positive literal. Since $\neg q$ is shown with better rank than q, NO is returned by the algorithm.

UNK : The only case not covered by the above is a global theory that entails neither q nor $\neg q$ classically and has as a prime implicate a positive clause containing q (if q is the positive literal). In this case, the status of q (and $\neg q$) is unknown according to the GCWA. In this case, our system and algorithm will do the following. Some peer whose local signature includes q will produce $\neg q$ with rank $\mathcal{C}$ according to its local CWA. The same or other peers will also produce the negation of each (positive) literal of the positive prime implicate clause containing q, and q will also be shown with rank $\mathcal{C}$. Since both q and $\neg q$ are shown with equal rank, the algorithm will return UNK. ■

In the centralized case of an individual knowledge base that contains only Horn clauses, the generalized closed-world assumption is equivalent to the regular closed-world assumption. The following corollary to Theorem 5.4 shows that this is also the case in the distributed case. Consider a PQAS that fullfills all the assumptions of the theorem, but additionally is constrained to only contain Horn clauses in all of it peers' local theories. Then the approach of assigning all local CWA consequences the priority $\mathcal{C}$ results in global closed-world reasoning, i.e. regular closed-world reasoning from the global theory, denoted in the corollary by $\models GCW$.

Corollary 5.5 (Global closed-world query answering from Horn theories)**.** *Let $\mathcal{P} = (P, G)$ be a PQAS with a satisfiable global knowledge base $\bigcup_{i=1}^{n} KB_i$ such that if for any two peers P_i and P_j, $L_{ij} = L_i \cap L_j$ is non-empty, $(i, j, L_{ij}) \in E$ and for all peers P_i, KB_i only contains Horn clauses and*

$Cons_i$ is the CWA entailment relation $\models_{CW}$ s.t. CWA-consequences are assigned the priority $\mathcal{C}$, which is lower than any peer's priority but higher than the priority represented by infinity. Answer_Query(q, P) of Algorithm 6 returns YES iff $\mathcal{P} \models_{GCW} q$ and NO iff $\mathcal{P} \not\models_{GCW} q$, where q is a literal.

Proof. By Definitions 5.1 and 5.2, the CWA and the GCWA are equivalent for all atoms except those that appear in positive prime implicates with two or more literals. Since all clauses in $\mathcal{P}$ are assumed to be Horn, the global theory has no positive prime implicates with more than one literal. Therefore, in a system only containing Horn local theories, making the GCWA wrt. the global theory is equivalent to making the CWA wrt. the global theory. Since our algorithm returns YES iff $\mathcal{P} \models_{GGC} q$ and NO iff $\mathcal{P} \not\models_{GGC} q$ by Theorem 5.4 and by the above observation about Horn theories and the equivalance of the CWA and the GCWA, it also returns YES iff $\mathcal{P} \models_{GCW} q$ and NO iff $\mathcal{P} \not\models_{GCW} q$. ■

The situation described in this subsection can be generalized along several dimensions. One constraint that could be relaxed is the connectivity of the PQAS. In this subsection, we have assumed that whenever two peers share a variable in their local signatures, then an edge between the two peers labeled by the variable exists in the connection graph. This is not always the case in general. Another constraint we have imposed on the PQAS of this subsection is that we required *every* peer to make the local closed-world assumption. In the general case, only some, not necessarily all, peers might produce local closed-world consequences. Furthermore, in a distributed setting, the desired global reasoning behavior might not always be best characterized in terms of entailment from the global theory as we have done in this subsection. The next subsection gives two informal examples of situations in which global closed-world reasoning is better characterized in terms of priorities directly rather than in terms of what can be entailed from the union of all local theories. In Section 5.3 we discuss such priority policies more generally.

5.2 Examples

In this subsection we consider two simple examples demonstrating how in different situations, closed-world consequences and their implications may have to be handled in different ways that cannot necessarily be characterized in terms of entailment from the global theory. For each example, we first characterize the scenario and then informally describe one possible way of achieving the desired global query answering behavior.

5.2.1 Online book shopping

Scenario Consider the scenario in Figure 9, which shows three book seller peers and a user peer wanting to buy a book. In this simplified case, all three book sellers are databases listing the books they carry. As such, they employ a local closed-world assumption and each bookseller peer returns $\neg x$ by closed-world entailment if it does not carry book x, regardless of whether another seller carries book x or not. Note that the book seller peers have different priorities (in parentheses), say due to user-perceived reliability or preference. Assume that two peers share all common vocabulary if there is an edge between them.

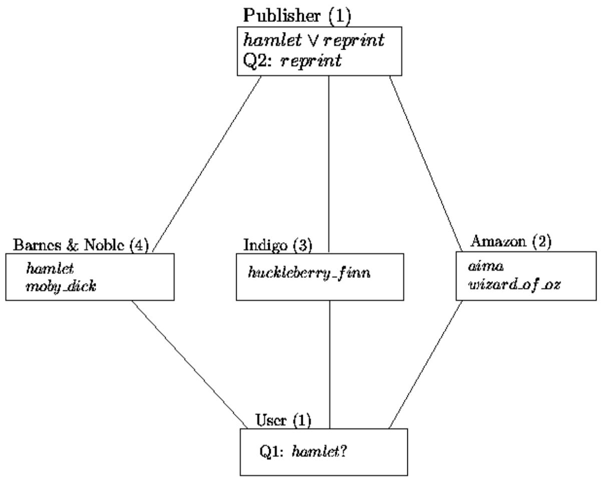

Figure 9: Online book shopping example

Consider now the query Q1 for *hamlet* asked by the *User* peer. Barnes & Noble will reply with *yes* while Indigo and Amazon will reply with *no*, corresponding to $\neg hamlet$ by closed-world entailment. Should the closed-world consequence $\neg hamlet$ override the positive consequence *hamlet* of the Barnes & Noble peer just because the latter has the lowest priority? Certainly not, since the user will want to buy the book from a lower-priority peer if a higher-priority peer does not carry it. In other words, we do not want to allow a positive consequence to be overridden by a closed-world consequence, no matter the priorities of the peers involved.

Let us now look at a another query, this time by the Publisher peer at the top of Figure 9, where one proposition depends on another one which may be produced by closed-world entailment. The Publisher peer infers the proposition *reprint* from $\neg$*hamlet*. It is assumed that the publisher needs to reprint the book once at least one bookstore runs out of stock. In other words, we wish to be able to prove the query *reprint* whenever at least one bookstore asserts $\neg$*hamlet* by closed-world entailment, even if other bookstores assert *hamlet* positively.

Note that to answer the second query, we made an inference from a CWA-induced literal ($\neg$*hamlet*) which was produced by some peers' local closed-world assumption (Indigo and Amazon) although another peer (Barnes & Noble) knows the positive literal explicitly. This may seem counterintuitive at first, but in some cases, as this example shows, this may be the desired behavior.

Solution We can achieve this behavior in a PQAS using our query answering algorithms of Section 4, again by fixing the priority of CWA consequences to be one universally lowest priority $\mathcal{C}$, say 10 for our example of Figure 9.

The first query, by the User peer, asked for the book *hamlet*. Our algorithm asks both *hamlet* and $\neg$*hamlet* as subqueries to decide which to believe. *hamlet* is proved with priority 4, while $\neg$*hamlet* is proved with priority $\mathcal{C} = 10$, which is lower (has greater numeric value) than the priorities of any of the peers. Since *hamlet* is proved with higher priority than $\neg$*hamlet*, it is believed. This is indeed the behavior we desire.

The second query, *reprint*, is asked at the Publisher peer, then negated and resolved with *hamlet* $\vee$ *reprint*. *hamlet* with priority 1 (from the Publisher peer) is then passed on to all three bookstore peers. It yields no empty clause consequence at the Barnes & Noble peer, but resolves with the CWA-induced $\neg$*hamlet* at the peers Indigo and Amazon, both times with priority 10. An empty clause constituting a reason for the original query is then passed back to the Publisher peer from both Indigo and Amazon with priority 10. As the negation of the query, $\neg$*reprint* cannot be proved at all in the system and is thus assigned priority ∞, the CWA-based reasons for *reprint* win and *reprint* is believed as a result. Again, this result corresponds to our desired outcome.

5.2.2 Weather forecast and flight delays

Scenario Consider now the scenario in Figure 10, in which the Airport peer is connected to two sources of current weather information: a news agency and a weather station. The news agency's weather knowledge is a time-delayed, but exact copy of that of the weather station so that both sources may temporarily disagree for short periods of time. Figure 10 is a snapshot of such a moment. Both peers make a local closed-world assumption and are thus able to produce CWA consequences. The weather station is a more trusted source (higher priority, in parentheses) than the news agency. The Airport peer knows that a flight is on time if the weather is not snowy. While the News Agency peer asserts *snowy*, the Weather Station peer asserts $\neg snowy$ by closed-world entailment. Even though it is a closed-world consequence, the airport would like to trust it over the explicit information from the news agency. As opposed to the previous example, in this example we want closed-world consequences from high-priority peers to override positive consequences from lower-priority peers.

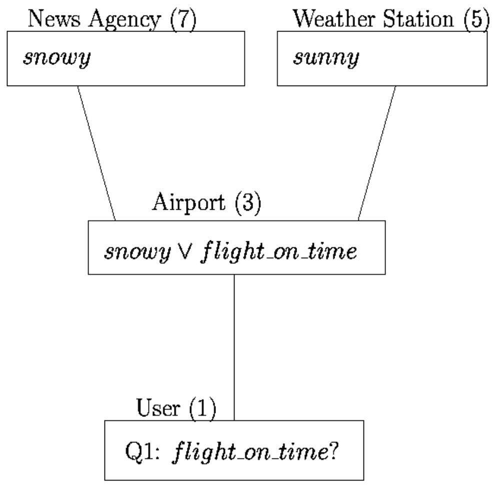

Figure 10: Weather forecast and flight delay example

Solution We can again achieve this behavior using our prioritized query answering algorithm, this time assigning local closed-world consequences a priority just lower than their peer of origin, but higher than any peer's priority that is less than the priority of the local CWA consequence's peer of origin. We would like a CWA consequence to have higher priority than the peers that have lower priority than the peer of origin so that the CWA consequence overrides the other peers' contradicting positive consequences. At the same time, we would like a CWA consequence to have lower priority than its peer of origin, so that positive consequences from that peer override contradicting CWA consequences. Recall that in the weather example the query $flight\_on\_time$ is asked. Given our prioritized query answering algorithm, again the query and its negation as posed as subqueries. The subquery $flight\_on\_time$ succeeds with priority 6 as the Weather Station peer locally produces $\neg snowy$ by closed-world entailment with just below its own priority (and higher than the News Agency peer, which has lower priority than the Weather Station peer). The $snowy$ consequence from the News Agency peer does not override the closed-world $\neg snowy$ consequence from the Weather Station peer as the latter has higher priority than the former and closed-world consequences have a priority just below that of their peer of origin. The subquery $\neg flight\_on\_time$ cannot be proved and thus receives priority ∞. Thus the original query is believed.

5.3 CWA Priority Policies

So far in this section we have formally defined and solved global generalized closed world reasoning in a restricted PQAS in which all peers make a local CWA. We have furthermore demonstrated by two examples that this is not the only meaningful way in which to globally reason with peers that make the CWA locally. In this subsection we discuss global reasoning in the presence of *some* (i.e. one or more, and potentially but not necessarily all) local CWA peers more generally.

Possibilities that are ignored in Section 5.1 include a setting in which only some, but not all, peers employ a local CWA reasoner. In addition, a better-priority peer might model its *known* negative information by the absence of positive information about the concerned literal and using a local CWA. Such negative information should be believed globally over conflicting positive information by another, but worse-priority peer. Our Definition 5.3 for global generalized closed-world entailment is not adequate anymore in the general case as it assumes that all peers make a local CWA and that local CWA consequence should always be overriden by positive knowledge from other peers, resulting in reasoning behavior that can be defined in terms of the global theory. We aim to relax both of these requirements.

So how should we define global reasoning in the new, general setting? The answer is to use priorities to characterize the desired global reasoning behavior of the system. Simply speaking, we always want the best-priority consequence to believed over any worse-priority conflicting information. This consequence may now very well have originated as a result of the local CWA of some peer. Several priority policies for local CWA consequences can be imagined and justified. We present a few of them here.

CWA Expert Peers One can think of a system of peers each of which has expert knowledge about a certain domain (a set of propositions). Each such peer may represent negative, but certain, information by leaving out the corresponding literal from the local KB and employing a local CWA. Since such a peer is still considered an expert with respect to that literal, its negation should be able to be used to prove other consequences with better priority than negations of such consequences from non-expert peers. This behavior can be achieved by setting the priorities of local CWA consequences to the priorities of the peer they originate from, i.e. by treating them just like regular local consequences. Note that a peer will only derive a locally non-entailed literal by the CWA if the corresponding variable is in its local language, so it is not the case that the best-priority expert peer will override any other peers' information with its CWA consequences. The weather station in the flight delay example of Section 5.2 illustrates this idea. The Weather Station peer is considered an expert in its domain of expertise (the propositions *snowy* and *sunny*), and CWA consequences in terms of these atoms override positive ones from the News Agency peer.

Layered Global CWA A special case of the previous CWA prioritization policy could be to have CWA consequences from a peer be overridden by positive consequences from peers with the same (or better) priority, but still have the CWA consequences override positive ones from worse-priority peers. In this case, the priority of a CWA consequence should be worse than the priority of the originating peer, yet better than those of peers with worse priority than the originating peer of the particular CWA consequence. If the smallest difference in priority among peers were ten, for example, it would suffice to give local CWA consequences a priority of one worse than the priority of the producing peer. Each set of peers of the same priority could then be seen as a layer in which “layer-wide” closed-world reasoning is done. For reasoning with local CWA consequences across layers, the consequences of the better-priority layers “win”, regardless of whether they are CWA or regular consequences.

Heterogeneous Systems The above two policy examples only discuss systems in which all peers make a local CWA. However, in real life most systems would likely be a mixture of peers that make the CWA locally and those that do not. Likewise, some of the peers that do make a local CWA may do so as part of their knowledge modeling paradigm (leave out known negative data and have it infered by the CWA) and those that list known negative data explicitely in the KB and only assume unknown data as negative to close their local worlds. Both types of peers would be handled by using different priorities for their respective CWA consequences. Such hybrid systems show the true strength of using priorities to handle local CWA consequences in the global setting—the designer of a particular PQAS, or those of its individual peers, can dictate how a local CWA consequence is handled or interpreted in the global setting by assigning them the appropriate priorities.

Many other specific policies besides the three mentioned above are thinkable. Having defined the global behavior of a PQAS that hosts some local CWA peers, we can simply apply our message-passing algorithm of Section 4.2 to compute query answers adhering to the given priority policy. The algorithm already answers queries according to whether the query or its negation can be shown with better priority and thus automatically handles any given prioritization scheme.

5.4 Discussion

In this section we formalized global generalized closed-world entailment for a small, restricted class of PQASs, namely those in which all consequences can propagate, all of whose peers make the local closed-world assumption, and the global theory of which is satisfiable. We characterized global generalized closed-world entailment in this setting in terms of generalized closed-world entailment from the global theory. We furthermore showed how query answering for single-literal queries according to global generalized closed-world entailment can be achieved using our algorithms from Section 4. We proved that our method is sound and complete for this case and discussed possible generalizations. In Section 5.2, we gave two examples demonstrating that this is not the only meaningful way to characterize global closed-world reasoning in a PQAS. In fact, different situations may require different treatment of local closed-world consequences. In Section 5.3 we discussed prioritization policies for local CWA consequences more generally. A further extension to the schemes provided in that subsection could be the conditional or context-specific attribution of priorities, although this may require meta reasoning. For example, a given peer may have high-priority knowledge about one domain, whereas its knowledge about another domain is less reliable. A thorough investigation of this question is outside the scope of this report and will be left for possible future research.

6 Implementation and Experiments

In order to evaluate our query answering algorithms and optimization techniques for them, a PQAS simulation was implemented. In this section, we briefly describe this implementation and the experiments it was used for. Finally, we report and analyze the results of these experiments.

6.1 Implementation

The simulation runs on a single machine and maintains one message queue for each simulated peer. Peers then take turns to process one message each. Three variations of the message-passing query answering algorithm have been implemented: the naive approach, a version that keeps track of reasons for the query and prunes away clauses and peers with worse priority than the currently best known reason, and a version that, in addition to pruning, employs the priority-ordering heuristic of Section 4.2.5. In the former two versions, the message queue is a simple first-in first-out (FIFO) queue that processes messages in the order they are received. In the latter version of the algorithm, a priority queue is employed which processes better-priority messages first. Messages with the same priority are processed in the order received. The priority of a message is the priority of the clause it represents (split-off literal for *forth* messages, empty clause for *back* messages); an exception are *prio* messages, which always have priority 0 so they get processed right away.

6.2 Experiments

In order to evaluate the effectiveness of both the pruning strategy and the priority-ordering heuristic, experiments involving randomly generated PQAS instances were executed.

6.2.1 Methodology

Each random PQAS had 4–20 peers with 4–8 clauses per peer, 1–6 shared propositions per peer, and a total of 8–96 distinct propositions. All three variations of the message-passing query answering algorithm were run on several hundred randomly generated PQAS instances, and for each PQAS and each variation of the algorithm, every proposition was asked as a query.

An attempt to answer a query was aborted after ten minutes (on a single machine). In most cases, the eventual answer had already been found by the time an algorithm gave up, but not yet been verified as the final answer by exhausting the (non-prunable) search space.

6.2.2 Results

Figures 11, 12, 13, and 14 show the experiment results. A total of 12,908 queries have been asked, but only those that could be solved within the alloted ten minutes by at least one version of the algorithm are considered in the figures. Also only non-trivial queries are shown in the graphs, i.e. those that took the naive algorithm at least a total of 300 messages to solve. The queries that took the naive algorithm less than 300 messages to solve took on average 52, 52, and 46 messages to answer for the naive, pruning, and ordering versions of the algorithm, respectively, so ignoring queries that are trivial for the naive algorithm does not hurt its evaluation in the comparison with the other two approaches.

The bar plot in Figure 11 shows the number of queries solved by each of the three versions of the algorithm within the alloted ten minutes. Clearly, the naive algorithm is outperformed by the one employing the pruning technique, which is in turn outperformed by the one additionally using the priority-ordering heuristic. Figure 12 compares the average total number of messages sent throughout the PQAS for proving a single query (or until giving up after ten minutes for unsolved problems). Notice the sharp reduction in messages that need to be sent between peers when employing pruning, and again when adding the ordering heuristic. Note that the bar for the naive algorithm also includes the number of messages sent for unfinished queries (given up on after ten minutes) which could be solved by the more sophisticated versions of the algorithm. Thus, the improvements of the pruning and ordering versions of the algorithm are actually understated in this figure. We chose the number of messages sent to answer a query as the performance measure as in a real-world distributed system, messages would be sent over a network, creating a bottleneck for the system.

Figure 13 plots the number of messages taken to solve a query (or in some cases for the naive algorithm, to give up) for all non-trivial queries. The top plot shows the number of messages for both the naive algorithm and the pruning version of the algorithm. The problem instances (queries) are sorted by the number of messages taken by the naive approach. The bottom

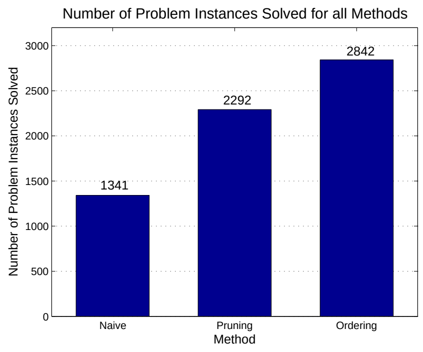

Figure 11: Number of problem instances solved by each method

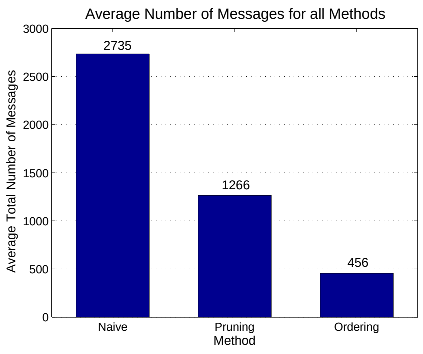

Figure 12: Average number of messages per problem instance until solved or given up

plot shows the number of messages for the pruning and the ordering heuristic version of the algorithm and is sorted by the number of messages taken by the pruning-only approach. Note that the pruning strategy offers significant improvement over the naive algorithm in many cases. Furthermore, adding the priority-ordering heuristic often significantly improves the performance again over the pruning-only method. Figure 14 is set up like Figure 13 but shows the difference in the number of messages taken for both pairs of algorithms. In particular, its top graph shows that pruning often saves no messages for simpler problem (further left) but saves more and more messages for harder ones (further right). The bottom plot shows again that, in a majority of cases, messages are saved by employing the priority-ordering heuristic.

Notice that in very few cases the pruning method takes slightly more messages than the naive algorithm (the bottom line crossing the top line in Figure 13 and small negative message savings in Figure 14). This situation occurs when no clauses, peers, or messages can be pruned away but the *prio* messages to update the priority of the currently best known reason (the variable *best* in Algorithms 8, 9, 10, and 11) are nevertheless sent. The naive algorithm does not have to send *prio* messages. Likewise, in a few degenerate cases, the priority-ordering version of the algorithm requires a somewhat larger number of messages than the pruning-only version. This may occur when a medium-priority message generating a reason and *best*

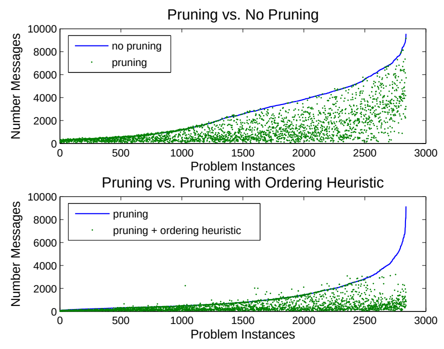

Figure 13: Number of messages until solved or given up for each problem instance

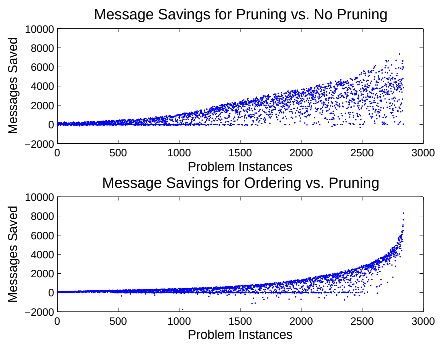

Figure 14: Number of messages saved due to pruning and ordering heuristic for each problem instance

update happens to be at the front of the FIFO queue in the pruning-only algorithm, but the heuristic version processes better-priority messages that lead to no reason first, potentially generating messages of bad priority before the pruning cutoff variable *best* is updated.)

7 Related Work

In this section we discuss some of the work related to the main themes of this report. We examine work in distributed reasoning, reasoning with inconsistencies in several different environments, trust models and trust aggregation, as well as work on global closed-world reasoning and look at similarities and differences to our work.

7.1 Distributed Reasoning

Several lines of work explore different aspects, uses, and settings of distributed reasoning.

P2P information management systems Perhaps the most straightforward way of making knowledge available in a distributed manner is by distributing databases in a peer-to-peer (P2P) fashion. In his PhD thesis, Zaihrayeu introduces query answering techniques for such P2P Information Management Systems [44]. A P2P Information Management System consists of either databases or ontology-based peers, with data and queries being translated and propagated between peers. While this is one of the first pieces of work on peer-to-peer reasoning, queries only refer to facts and no expressive logical language for full deductive reasoning is supported.

Distributed FOL and DDL Ghidini and Serafini developed a logic for reasoning about a set of knowledge bases, each in its own first-order language [19]. The authors give the syntax and semantics for their Distributed First-Order Logic (DFOL) and define entailment relations and a calculus for reasoning in it. DFOL is used to reason *about* distributed reasoning and cannot be used in an actual distributed reasoning system.

Borgida and Serafini also defined and investigated distributed descriptions logics (DDL), which support complex mappings between the domains of local knowledge sources via bridge rules [18]. These mappings enable the deduction of new information such as subsumption relationships between concepts of different local knowledge sources. Serafini and Tamilin furthermore developed a tableaux-based reasoning procedure for DDL and the DRAGO distributed ontology reasoning system based on this procedure [38, 39].

Partition-based logical reasoning We already discussed the partition-based reasoning work by Amir and colleagues in Section 2. In this line of work, large knowledge bases are decomposed into partitions of related knowledge [5, 6, 33]. Partitions are connected in a tree graph and message-passing procedures to reason with them are developed. This work is not motivated by achieving global reasoning in distributed systems but by gaining efficiency for individual, centralized knowledge bases by splitting them up into loosely connected partitions. Thus, the procedure has full control of the topology of the graph, which is not true for truly distributed reasoning settings. Nevertheless, the algorithms provided in this line of work can be used in a distributed setting, provided the graph topology is a tree. This, however, is a major restriction for realistic systems and prohibits their use in a real-world setting.

Peer-to-peer inference systems We have also thoroughly reviewed peer-to-peer inference systems (P2PISs) in Section 2. P2PISs are the first fully distributed propositional reasoning systems [2, 3, 4]. In this work, the authors assume that a P2PIS is globally consistent and provide a fully distributed consequence-finding algorithm for it. Although part of our work is based on this algorithm, the specific problem we address is query-answering rather than consequence-finding. Most importantly, our approach handles systems that consist of individually consistent peer knowledge bases that may be inconsistent when combined. Furthermore, our method also handles prioritized local knowledge bases, providing for the foundations of a mechanism to address trust or reliability issues between peers and resolving ambiguous inconsistencies. A recent paper in the P2PIS line of work attempts to handle inconsistencies but only does so in a simple manner and without priorities ([20], see the next subsection).

7.2 Reasoning with Inconsistencies

Reasoning with inconsistencies has been investigated in the database and knowledge representation communities for many years. Some have suggested to repair inconsistent databases by removing or ignoring facts or knowledge causing inconsistencies [8, 9]. Given a repaired database or knowledge base, classical query answering or reasoning can be performed to compute meaningful results. However, since some knowledge is deleted, information is arguably being lost. The following lines of work attempt to reason with inconsistent knowledge rather than fixing it beforehand.

Inconsistent DBs and integrated data sources Arenas and Bertossi et al. introduced mechanisms to reason with inconsistent knowledge rather than modifying it first as we have already seen in Section 2.2.1. In this work, the authors use query rewriting techniques to express an original query as a rewritten query whose regular answers are the consistent answers to the original query [7, 14, 17, 16]. This work treats query answering from inconsistent knowledge in a restricted subset of ground first-order logic applied to relational databases. A database instance is a collection of ground literals and the integrity constraints over them are first-order formulas. Bertossi and Bravo build on this work to describe peer-to-peer networks of databases that account for trust between them to achieve consistent query answering given data exchange constraints that may cause inconsistencies between the individual peers [15].

Handling inconsistent knowledge bases Besides in the database community, reasoning with inconsistencies has also received much attention in a centralized, but full logical reasoning setting. We have already discussed the work of Benferhat and colleagues which defines and compares several entailment relations for reasoning with inconsistent knowledge bases in Section 2.2.2 [11, 13, 12]. Elvang-Gøransson and Hunter previously discussed some of the entailment relations (involving arguments) investigated in this line of work [26]. The entailment relation that stood out most for our purposes was argued entailment for prioritized knowledge bases [12]. While argued entailment allows for three or more consequences to be mutually inconsistent, no two consequences can be inconsistent. In particular, argued entailment does not allow both a formula and its negation to be a consequence at the same time. As in this work we are concerned with answering one query at a time, but having to decide whether to believe the query or its negation, this formulation of entailment in inconsistent knowledge bases suits our purpose well. In particular, it enabled us to achieve more complex or coherent query answering than the work on inconsistent peer-to-peer inference systems, which we shall briefly talk about next.

Inconsistent peer-to-peer inference systems Prior to the present report, Chatalic et al. have introduced the first propositional peer-to-peer reasoning framework for reasoning with inconsistencies [20]. This work is an extension of the peer-to-peer inference systems described in Section 7.1 and we already introduced this work in Section 2.2.3. The authors develop an algorithm to compute and store *nogoods*, which are the mapping clauses between peers that may cause inconsistencies, whenever a new peer is added. Given the nogoods, another algorithm computes *well-founded consequences* from a query literal. A well-founded consequence is one that depends only on a consistent subset of the global theory. Note that given this definition, both a formula and its negation may be derived as a well-founded consequence. Our approach introduced in this report relies on argued entailment [12] and therefore does not suffer from this issue.

Distributed truth maintenance systems Truth maintenance systems (TMS) were first introduced by Doyle in order to keep a knowledge base consistent when new knowledge is added by reasoning not only about conclusions but also the reasons for believing them [25]. The first TMSs were justification-based TMSs (JTMS) that keep track of justifications for currently believed information (information that is *in*) as well as previously believed information that may be believed again at a later point (information that is *out*). Assumption-based TMSs (ATMS) were introduced by de Kleer to keep track of assumptions and their consequences in order to represent severals possible states of knowledge at the same time [24]. Huhns and Bridgeland introduced distributed truth maintenance systems (DTMS) [31] to be used in their multi-agent TMS developed in [32]. The authors introduce a multi-agent setting in which agents host local knowledge bases and are able to send information to and receive it from other agents. A multi-agent TMS is based on the JTMS and annotates pieces of information as *internal*, *external*, or *out* to maintain various levels of local and global consistency based on the justifications for these pieces of information. Internal information is believed at the moment and local to an individual agent. External information is also currently believed but shared between multiple agents. Out information is not currently believed. Since the multi-agent TMS is based on the JTMS, only one state of knowledge can be represented, along with its justifications. This state of knowledge is required to be consistent according to the various degrees of consistencies defined by the authors. Agents can negotiate with each other in order to decide which knowledge can be kept and which will be revised.

7.3 Trust

Our approach to assign priorities to peers and use them to resolve conflicting information is closely related to the concept of trust. However, while in our approach priorities are absolute and fixed, the concept of trust would be relative to a peer. I.e. while peer P_1 may have a universal priority of 10, two different peers might trust P_1 differently, say with values of 12 and 8.

Trust and similar concepts have received some attention in both peer-to-peer file sharing systems as well as for electronic commerce. Gupta et al. introduced a reputation system for peer-to-peer file sharing networks [28]. Aberer and Despotovic developed a method for the management and computation of trust in peer-to-peer information systems and provide an algorithm implementing it [1]. Yu and Singh introduced mechanisms for reputation management in electronic commerce [42]. In their approach, reputation values are aggregated based on the Dempster-Shafter theory of evidence in a distributed fashion [42, 40]. Partly building on this work, Wang and Singh developed trust aggregation mechanisms for distributed peers in [41].

Another related area of work concerns knowledge provenance, or tracking the origin of pieces of knowledge in order to determine their reliability. Silva et al. describe a knowledge provenance infrastructure to integrate Web applications that require such provenance information [22]. Huang and Fox capture a model for reasoning about knowledge provenance in first-order logic and provide mechanisms to deal with changes in provenance of time as well as uncertainty in knowledge provenance information [29, 30].

Much of the work on trust in systems of multiple information sources involves numeric or at least comparable measurements of trust, certainty, reliability, or priority. To this end, our work introduces a technique to accumulate priority information while proving or disproving the query under investigation and using this information in the query-answering process.

7.4 Global closed-world reasoning

In our work we have addressed to some extent the problem of reasoning with local closed-world reasoning peers that are connected to other peers in a decentralized setting such as the Semantic Web. This problem has already received some attention for multiple connected databases, logic programs, and description logic reasoners. Cortés-Calabuig et al. formalized the concept of a local closed-world assumption of data-sources [21]. In their approach, meta-level first-order logic expressions are used to describe local closed-world assumptions (LCWAs) of individual data sources. A LCWA specifies about which atoms a particular knowledge source has complete information. Such a LCWA can then, along with the actual data in the corresponding knowledge source, be used to make negation-as-failure inferences that will hold globally (i.e. in conjunction with other data sources). Damásio et al. have addressed the issue of open and closed-world reasoning with rule bases or logic programs for the Semantic Web [23]. In this work, the authors use explicit logic program transformations to integrate a local reasoner with other reasoners into the global setting. Grimm and Motik use autoepistemic description logics to achieve closed-world reasoning in an open-world setting [27]. Description logics are commonly used and researched for use in reasoning with ontologies in the Semantic Web. The work in this report adds to this area by identifying two policies for treating local closed-world consequences in a global setting as well as demonstrating how global closed-world reasoning with local closed-world reasoners can be achieved in a restricted propositional setting.

8 Conclusion

Peer-to-peer reasoning has started to receive much attention due to the emergence of the Semantic Web. While a consequence-finding algorithm for the consistent case exists, inconsistent systems have only been addressed in a limited way. In this report, we have focused on solving the query-answering problem and addressing inconsistencies between peers. We have introduced priorities into this setting to be optionally used to either resolve conflicts in inconsistent systems or to find the best reason for a query rather than just any reason. The work with priorities is closely related to the concepts of trust and reliability, an extension to which is one possible area of future work.

8.1 Summary of Contributions

Formalization of peer-to-peer query answering systems We defined peer-to-peer query answering systems (PQASs) as collections of peers with local knowledge bases, reasoners, and their connectivity to and shared variables with other peers in the system. We made a distinction between $\mathcal{P}$-consistent and $\mathcal{P}$-inconsistent systems and defined distributed entailment relations for both.

Handling of priorities Furthermore, we provided a framework to optionally specify priorities for each peer. Priorities helped us in both resolving conflicts in $\mathcal{P}$-inconsistent peer-to-peer query answering systems as well as in finding the best reason for believing a query in terms of priorities and ranks. A best-rank reason could be interpreted as the most reliable one if priorities represent reliability. We provided for the aggregation of priorities in this distributed setting to establish a notion of ranks of reasons from the priorities of individual clauses.

Algorithms for inconsistency-tolerant query-answering In order to solve the query answering problem in the PQAS setting, we adapted and extended existing consequence-finding algorithms to form recursive and message-passing algorithms for query answering in both $\mathcal{P}$-consistent and $\mathcal{P}$-inconsistent systems. We established the soundness and completeness of the algorithms with respect to distributed entailment. We furthermore developed an admissible heuristic for more efficiently finding the best-rank reason for a query and conducted experiments to empirically demonstrate its effectiveness.

Global closed-world reasoning Keeping in mind that in the Semantic Web, individual closed-world reasoners may participate in the global reasoning process, we identified two examples of global reasoning in which local closed-world consequences need to be handles differently. We furthermore demonstrated how our techniques can be used to achieve generalized closed-world query answering from the global theory in a restricted case of PQASs with all local closed-world reasoners. We discussed in general terms how different priority policies can be used to achieve different types of global behavior in the presence of local closed-world reasoners.

8.2 Limitations and Future Work

Query types As discussed in Section 4.3.2, our algorithms are not complete for answering disjunctive queries. For inconsistent systems, this means that query answering is only complete for single-literal queries as the negation of each query has to be asked in addition to the query itself and the negation of a conjunctive query is a disjunctive query. This is clearly a limitation for many practical purposes and opens an important area of potential future work.

Priorities and trust In our approach, priorities are fixed for each peer in the system. At least two possible relaxations of this constraint come to mind. Firstly, since priorities are fixed and not relative to the perspective of each peer, our priorities framework and algorithms cannot be used as is to model trust between peers. In order to use the setup for the computation of trust values and most-trusted query answers, priorities would thus have to be dependent on the perspective of each individual peer. One way of doing so could involve transmitting the trust value in a peer along with a consequence or query. This and other approaches as well as how to avoid deception by malicious peers could be further explored in the future.

Secondly, our approach only supports peers whose knowledge bases are homogeneous in terms of the priorities of their clauses. It may be appropriate to distinguish between different priorities within an individual peer. This may be achieved either by splitting each peer into as many logical "subpeers" as there are different-priority clauses in its KB or by adapting our algorithms to handle single KBs with heterogeneous priorities.

Furthermore, we assumed in this work that priorities are given as comparable values from a totally ordered set and aggregated using the *max* function. In the more general case, other aggregation schemes are possible. For a simple example, the priority values of formulas (for modeling trust) could be propagated by multiplication with the trust values of each peer they travel through. If priority values are numeric values between zero and one, this would mean that the priority aggregation scheme is still monotonically decreasing with more and more peers involved in the aggregation. More generally, non-monotonic aggregation schemes are also thinkable, although we would then not be able to prune the search tree based on the current ranks for formula sets that may evolve into reasons for believing a query without losing completeness.

Global closed-world reasoning We gave two examples of how different policies on global closed-world reasoning might look like and discussed different possible priority policies that result in different kinds of global behavior in the presence of local closed-world reasoners. However, we have not formally specified the requirements for global closed-world reasoning with local closed-world reasoners in the general case. We have furthermore formally demonstrated how global closed-world reasoning can be achieved using our techniques in a restricted case and under one specific policy (to aim to achieve generalized closed-world reasoning from the global theory). In an interesting area of future work, our observations could be formalized and the different policies for global closed-world reasoning formally explored. It could be investigated whether algorithms for query answering exist given a general formal framework handling arbitrary policies for global closed-world reasoning.

Peer languages and mappings Perhaps the biggest deterrent to being able to use the present framework in a real-world Semantic Web setting is the existence of a single, global language. In any real-world application, individual peers would use different signatures, and mappings would have to be used to relate variables between a peer and its neighbors. While we consciously ignored this matter of semantic integration in order to concentrate on the algorithmic issues in dealing with priorities and inconsistencies, it would be an interesting and useful area of future work to extend our framework to include separate local peer languages and mappings between them.

Expressiveness of peer languages Lastly, our framework exclusively treats propositional logic peer-to-peer reasoners. For the benefit of more expressiveness, it should not be too hard to extend the framework and algorithms to handle the first-order case or a subset thereof, such as description logics, which enjoy high popularity in the Semantic Web community.

References

[1] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, pages 310–317. ACM Press, 2001.

[2] Philippe Adjiman, Philippe Chatalic, Francois Goasdou, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting. In *Proceedings of European Conference on Artificial Intelligence (ECAI 2004)*, pages 945–946, 2004.

[3] Philippe Adjiman, Philippe Chatalic, Francois Goasdou, Marie-Christine Rousset, and Laurent Simon. Scalability study of peer-to-peer consequence finding. In *Proceedings of IJCAI 2005 (International Joint Conference on Artificial Intelligence)*, 2005.

[4] Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *Journal of Artificial Intelligence Research*, 25:269–314, 2006.

[5] E. Amir and S. McIlraith. Partition-based logical reasoning. In *Proc of The 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 2000.

[6] E. Amir and S. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 2000.

[7] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *ACM PODS'99*, 1999.

[8] O. Arieli. An Algorithmic Approach to Recover Inconsistent Knowledge-bases. In M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. Moniz Pereira, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000, Málaga, Spain, September/October 2000. Proceedings*, pages 148–162. Springer Verlag, 2000.

[9] O. Arieli, M. Denecker, B. Van Nuffelen, and M. Bruynooghe. Repairing inconsistent databases: A model-theoretic approach. In *Proceedings PCL 2002, Federate Logic Conference (FLoC)*, 2002.

[10] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

[11] S. Benferhat, D. Dubois, and H. Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: a comparative study part 1: The flat case. *Studia Logica*, 58:17–45, 1997.

[12] S. Benferhat, D. Dubois, and H. Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: a comparative study part 2: the prioritized case. In E. Orłowska, editor, *Logic at Work: Essays Dedicated to the Memory of Helen Rasiowa*, volume 24, pages 437–511. Physica-Verlag, 1999.

[13] Salem Benferhat, Didier Dubois, and Henri Prade. An overview of inconsistency-tolerant inferences in prioritized knowledge bases . In D. Dubois, H. Prade, and E.P. Klement, editors, *Fuzzy Sets, Logic and Reasoning about Knowledge* , volume 15 of *Applied Logic Series*, pages 395–417. Kluwer, Dordrecht, Pays-Bas, 1999.

[14] L. Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76, June 2006.

[15] L. Bertossi and L. Bravo. Query answering in peer-to-peer data exchange systems. In *EDBT Workshop on Peer-to-Peer Computing and Databases*, pages 476–485, 2004.

[16] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In G. Saake J. Chomicki and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*. Springer, 2003.

[17] Leopoldo E. Bertossi, Jan Chomicki, Alvaro Cortés, and Claudio Gutiérrez. Consistent answers from integrated data sources. In *FQAS '02: Proceedings of the 5th International Conference on Flexible Query Answering Systems*, pages 71–85, London, UK, 2002. Springer-Verlag.

[18] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.

[19] Ghidini C. and Serafini L. Distributed first order logics. In D.M. Gabbay and M. De Rijke, editors, *Frontiers of Combining Systems 2*, number 7 in Studies in Logic and Computation, pages 121–139. Research Studies Press Ltd. Baldock, Hertfordshire, England, UK, 2000.

[20] Philippe Chatalic, Gia-Hien Nguyen, and Marie-Christine Rousset. Reasoning with inconsistencies in propositional peer-to-peer inference systems. In *Proceedings of ECAI 2006 (European Conference on Artificial Intelligence)*, pages 352–357, 2006.

[21] Alvaro Cortés-Calabuig, Marc Denecker, Ofer Arieli, Bert Van Nuffelen, and Maurice Bruynooghe. On the local closed-world assumption of data-sources. In *BNAIC*, pages 333–334, 2005.

[22] P. da Silva, D. McGuinness, and R. McCool. Knowledge provenance infrastructure, 2003.

[23] Carlos Viegas Damásio, Anastasia Analyti, Grigoris Antoniou, and Gerd Wagner. Supporting open and closed world reasoning on the web. In *PPSWR*, pages 149–163, 2006.

[24] J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence*, 28(2):127–162, 1986.

[25] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[26] Morten Elvang-Gøransson and Anthony Hunter. Argumentative logics: reasoning with classically inconsistent information. *Data and Knowledge Engineering*, 16(2):125–145, 1995.

[27] Stephan Grimm and Boris Motik. Closed world reasoning in the semantic web through epistemic operators. In *Proceedings of the Workshop OWL - Experiences and Directions*, 2005.

[28] Minaxi Gupta, Paul Judge, and Mostafa Ammar. A reputation system for peer-to-peer networks. In *NOSSDAV '03: Proceedings of the 13th*

international workshop on Network and operating systems support for digital audio and video, pages 144–152, New York, NY, USA, 2003. ACM Press.

[29] Jingwei Huang and Mark S. Fox. Dynamic knowledge provenance. Technical report, Enterprise Integration Laboratory, University of Toronto, 2003.

[30] Jingwei Huang and Mark S. Fox. Uncertainty in knowledge provenance. In *ESWS*, pages 372–387, 2004.

[31] Michael N. Huhns and David M. Bridgeland. Distributed truth maintenance. In S. M. Dean, editor, *Cooperating Knowledge Based Systems*, pages 133–147. Springer-Verlag, 1990.

[32] Michael N. Huhns and David M. Bridgeland. Multiagent truth maintenance. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1437–1445, 1991.

[33] Bill MacCartney, Sheila A. McIlraith, Eyal Amir, and Tomas Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, August 2003.

[34] J. Minker. On indefinite databases and the closed world assumption. In *Lecture Notes in Computer Science*, volume 138, pages 292–308. Springer, Berlin, 1982.

[35] Raymond Reiter. On closed world data bases. In Herve Gallaire and Jack Minker, editors, *Logic and data bases*, pages 55–76. Plenum Press, New York, NY, 1978.

[36] N. Rescher and R. Manor. On inference from inconsistent premises. *Theory and Decision*, 1:179–217, 1970.

[37] Marie-Christine Rousset. Small can be beautiful in the semantic web. In *Proceedings of International Semantic Web Conference (ISWC 2004)*, pages 6–16, 2004.

[38] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. Technical Report T04-12-05, ITC-irst, December 2004.

[39] L. Serafini and A. Tamilin. Local tableaux for reasoning in distributed description logics. In V. Haarslev and R.Moeller, editors, *Proceedings of the 2004 Intl. Workshop on Description Logics (DL2004)*, pages 100–109, 2004.

[40] G. A. Shafer. *Mathematical Theory of Evidence.* Princeton University Press, Princeton, N.J., 1976.

[41] Yonghong Wang and Munindar P. Singh. Trust representation and aggregation in a distributed agent system. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[42] Bin Yu and Munidar P. Singh. Distributed reputation management for electronic commerce. *Computational Intelligence*, 18(4):535–549, 2002.

[43] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 100(supp.):9–34, 1999.

[44] Ilya Zaihrayeu. *Towards Peer-to-Peer Information Management Systems.* PhD thesis, University of Trento, March 2006.